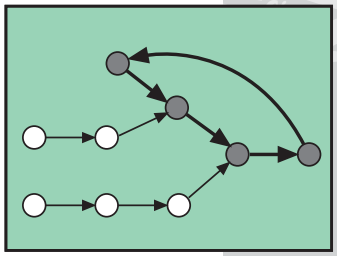
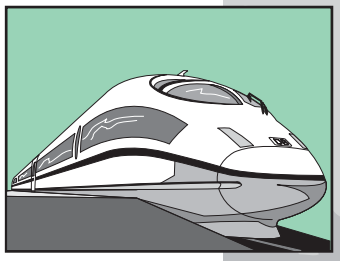
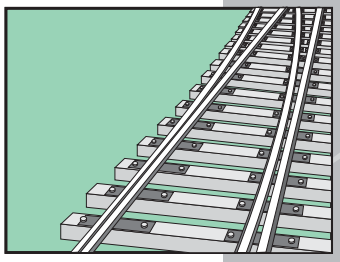
Institut für Eisenbahnwesen
und Verkehrssicherung

IfEV
TU Braunschweig







SAEID ARABESTANI

Formal verifizierbare objektorientierte Systemspezifikationen mit UML für Eisenbahnsicherungssysteme

Schriftenreihe • Heft **67** • 2005

S C H R I F T E N R E I H E

INSTITUT FÜR EISENBAHNWESEN UND VERKEHRSSICHERUNG DER
TECHNISCHEN UNIVERSITÄT BRAUNSCHWEIG

Herausgeber:
Prof. Dr.-Ing. Jörn Pachl

Formal verifizierbare objektorientierte Systemspezifikationen mit UML für Eisenbahnsicherungssysteme

von
Dipl.-Inform. Saeid Arabestani
aus Teheran

Heft 67
2005

Institut für Eisenbahnwesen und Verkehrssicherung
Technische Universität Braunschweig
Prof. Dr.-Ing. Jörn Pachl

Pockelsstraße 3
38106 Braunschweig

Telefon: + 49 - (0) 531 - 391 - 3380
Telefax: + 49 - (0) 531 - 391 - 5955

Vom Fachbereich Bauingenieurwesen der Technischen Universität Carolo Wilhelmina zu Braunschweig zur Erlangung des Grades eines Doktor-Ingenieurs (Dr.-Ing.) genehmigte Dissertation

Eingereicht am	22. April 2005
Mündliche Prüfung am	19. Juli 2005

Berichter	Prof. Dr.-Ing. Jörn Pachl
	Prof. Dr. Hans-Dieter Ehrich

ISSN 0721 - 7137
ISBN 3 - 923325 - 67 - 3

تقدیم به مادرم
به خانم خدیجه کفایتی

Für meine Mutter
Frau Khadijeh Kefayati

در ره عشق نشد کس به یقین محرم راز
هر کسی بر حسب فهم گمانی دارد

حافظ

Vorwort

Bei der Entwicklung und Zulassung von Systemen der Eisenbahnsicherungstechnik kommt der Erstellung der Anforderungsspezifikation eine herausgehobene Bedeutung zu. Die Anforderungsspezifikation beschreibt aus Sicht des künftigen Anwenders alle funktionalen Anforderungen, die das System erfüllen muss. Anforderungsspezifikationen für Eisenbahnsicherungsanlagen werden heute noch immer in Form einer verbalen Beschreibung, ggf. ergänzt durch Illustrationen, erstellt. Mit dem Übergang zu rechnergestützten Sicherungsanlagen wurde eine Komplexität erreicht, für die dieses traditionelle Verfahren nicht mehr zielführend ist. Trotzdem ist es bisher nicht gelungen, ein auf formalen Beschreibungsmethoden basiertes Verfahren zu entwickeln. Der Grund dafür liegt vordergründig in der fehlenden Anwenderfreundlichkeit formallogischer Beschreibungsmittel. Als Alternative existieren die semiformalen Methoden, die durch grafische Beschreibungsmittel dem Verständnis von Praktikern wesentlich besser zugänglich sind und zudem durch die Darstellung von Strukturen und Abläufen in bildlicher Form dem klassischen Denken des Ingenieurs sehr nahe kommen. Die Anwendung semiformaler Methoden erzwingt eine saubere Strukturierung und vermeidet somit eine Reihe der bei einer rein verbalen Anforderungsspezifikation möglichen Fehler. Diesem aus Anwendersicht an sich überzeugenden Vorteil steht jedoch der entscheidende Nachteil der fehlenden Formalisierung gegenüber, wodurch die Möglichkeit einer Verifizierbarkeit der Anforderungsspezifikation nicht besteht. Wegen dieses Nachteils rechnet sich der nicht zu unterschätzende Aufwand zum Erlernen einer semiformalen Methode

aus Anwendersicht kaum. Der entscheidende Qualitätssprung wäre der Schritt zur formalen Beschreibung, den diese Verfahren allein nicht leisten.

In der vorliegenden Dissertation wird nun erstmalig eine Methode vorgestellt, die anwenderfreundliche semiformale Beschreibungsmittel mit formalen Definitionen verbindet und durch diese Formalisierung die Verifikation von Anforderungsspezifikationen ermöglicht. Durch die Möglichkeit, nach einer Transformation in temporallogische Ausdrücke die Erfüllung vorgegebener Prüfbedingungen durch Model-Checking mit der Aussageschärfe eines mathematischen Beweises zu prüfen, wird ein bisher unterreichter Grad der Fehleroffenbarung in Anforderungsspezifikationen erreicht. Dies stellt für die Anforderungsspezifikation von Systemen der Eisenbahnsicherungstechnik eine vollkommen neue Qualität dar.

Braunschweig, im August 2005

Prof. Dr.-Ing. Jörn Pachl

Inhaltsverzeichnis

Abstract

Regarding the development, the operation and the certification of railway signalling systems there are three groups of engineers involved and every one of them has different tasks. First of all, one group contains the client who usually also is the future railway infrastructure manager. After having discussed the necessary requirements of the system to be developed with experts the client writes them down and illustrates them with graphics. Furthermore, there is the contractor who will develop the system and give a proof of safety. Ideally, he will try to prove the system's safety, correctness and consistency during every development phase. The third party is the safety authority which is in charge of verifying the client's requirements as well as the safety case. Moreover, the safety authority is also concerned with the certification of the developed system.

The textual description of the system requirements specification is the basis for further development. Although thoroughly thought through the description may still be incomplete, inconsistent or subject to ambiguities. Verifying the consistency and compliance with safety requirements of textual system requirements specifications is extremely difficult.

In order to increase the precision and verifiability of system requirements specifications the CENELEC-Standards for railway signalling systems demand the use of formal methods because they possess unequivocal syntax and semantics. However, not all of

the groups mentioned in the beginning of this abstract are able to handle formal methods (equally well).

On the contrary, the semi-formal specification language Unified Modeling Language (UML) has spread widely in engineering and industrial applications. Due to its graphical feature users find it easy to learn and comprehensible. This characteristic is the main reason for UML's constantly growing popularity among engineering applications.

The means of description in UML lack unequivocal and formal semantics. On the one hand this complicates a sufficiently precise description of the facts of the application area. On the other hand it makes a formal verification with the goal of proving the correctness and safety compliance of the system safety requirements more difficult.

In my paper I present formal-based graphical means of description for the specification and verification of railway signalling systems which are based on formal methods. These means of description are a selection of notation elements of UML and they allow the precise modelling of the system structure as well as the dynamical behaviour of complex railway signalling systems. Due to their graphical nature the selected notation elements are user-friendly and easy to learn. These characteristics are demonstrated using the example of modelling the single-track level crossing in a radio-based operating system (FFB developed by DB-AG).

By using the introduced formal semantics for the employed notation elements a formal verification of the system requirements specification is possible. This verification is based on the technique of Model-Checking where the checking of the system requirements specification is conducted automatically and computer-aided.

As a result I provide means of description for developing formally verifiable system requirements specifications. These means can be handled by engineers responsible for railway signalling systems and they also meet the CENELEC-Standards' demands on increased precision and verifiability.

Saeid Arabestani

1 Einleitung

Eine wachsende Einbindung von Rechnernetzen und modernen Kommunikationssystemen in die Systeme der Leit- und Sicherungstechnik im Eisenbahnwesen führt dazu, dass die technischen Sicherungsanlagen immer komplexer werden. Für die Sicherungssysteme ergeben sich durch die Einbindung moderner Techniken und durch die neuen europäischen CENELEC-Normen [EN50126, EN50128, EN50129] noch besondere Anforderungen, die aus dem Entwicklungs- und Zulassungsprozess erwachsen.

Der erste Schritt bei der Entwicklung von Eisenbahnsystemen besteht darin, eine Systemanforderungsspezifikation als Ausgangspunkt für eine Realisierung des Systems zu verfassen. Die Entwicklung der Systemanforderungsspezifikation ist die Aufgabe des Auftraggebers, der in der Regel auch der spätere Systembetreiber ist. Nach den CENELEC-Normen ist er auch verantwortlich für die Durchführung einer Risikoanalyse für das neue System. Dabei hat er auf Basis der Systemanforderungsspezifikation mittels der Risikoanalyse die zulässigen Raten für die betrieblichen Gefährdungen zu bestimmen. Die Systemanforderungsspezifikation und die anschließende Risikoanalyse bedürfen einer Zusicherung durch die zuständige Aufsichtsbehörde. Die Aufsichtsbehörde überprüft die Konformität der Systemanforderungsspezifikation mit den relevanten Gesetzen und Vorschriften und mit den Anforderungen an die Sicherheit. Sie erteilt für die Systemanforderungsspezifikation eine Zusicherung bei positivem Prüfergebnis. Daraufhin kann der Auftraggeber die Systemanforderungsspezifikation für die Auftragsvergabe zur Systementwicklung verwenden.

Der Entwickler soll parallel zur Entwicklung des Systems noch den Sicherheitsnachweis führen. Im Idealfall versucht er, in jeder Entwicklungsphase die Sicherheitskonformität, Korrektheit und Konsistenz des Entwicklungsstands gegenüber der Systemanforderungsspezifikation bzw. der vorangehenden Entwicklungsphase zu zeigen. Ein Teil des Sicherheitsnachweises ist die Durchführung einer Gefährdungsanalyse, in der gezeigt werden soll, dass von dem entworfenen bzw. entwickelten System die vorgegebenen Gefährdungsraten eingehalten werden. Die Aufsichtsbehörde überprüft den Sicherheitsnachweis. Sie ist sowohl für die Prüfung der von dem Auftraggeber gestellten Systemanforderungsspezifikation als auch für die Überprüfung des von dem Entwickler vorgelegten Sicherheitsnachweises und anschließend für die Zulassung des entwickelten Systems zuständig.

Somit bildet die Systemanforderungsspezifikation die Basis für alle weiteren Tätigkeiten. Ihr kommt eine besondere Bedeutung im gesamten Entwicklungs- und Zulassungsprozess im Eisenbahnwesen zu.

Heute werden Systemanforderungsspezifikationen auch für sicherheitsrelevante Systeme meist noch in natürlicher Sprache formuliert, die durch grafische Darstellungen ergänzt werden. Die natürlich sprachliche Beschreibung wird in vertrauter Fachsprache verfasst und ist daher für die Eisenbahningenieure leicht verständlich. Sie ist jedoch trotz sorgfältiger Bearbeitung sehr schwer und nur mit hohem Aufwand auf Vollständigkeit, Konsistenz, Eindeutigkeit und Sicherheitskonformität überprüfbar. In einer derartigen Systemanforderungsspezifikation ist die Beschreibung der einzelnen Prozesse trotz mehrfacher jedoch notwendiger Wiederholungen, die ihrerseits den Umfang der Systemanforderungsspezifikation steigern, in der Regel nicht vollständig, nicht widerspruchsfrei und bedarf mehrfacher Rücksprache mit dem Auftraggeber. Ein weiterer Nachteil bei einer natürlichsprachlich verfassten Systemanforderungsspezifikation besteht darin, dass wegen fehlender formaler Fundierung Mehrdeutigkeiten unvermeidbar sind. Dieser Nachteil erschwert auch die Nachvollziehbarkeit bei komplexen Aufgabenstellungen. Ferner sind diese Systemanforderungsspezifikationen schwer erweiterbar und notwendige Korrekturen sind meist mit einem enormen Aufwand verbunden.

Um diesen Problemen entgegenzuwirken, empfehlen die neuen CENELEC-Normen für die sicherheitsrelevante Eisenbahnsignaltechnik den Einsatz von formalen Methoden für die Entwicklung von Systemen der Leit- und Sicherungstechnik. Diese Methoden sollen dazu beitragen, dass die Darstellung der Sachverhalte genügend präzise ist und der Aufwand zur Durchführung der notwendigen Überprüfungen reduziert wird. Die Forderung nach Einsatz formaler Methoden ist allerdings ohne konkrete und praktikable Anleitungen für die Anwendung dieser Methoden.

Die vorliegende Arbeit hat zum Ziel, formal fundierte Beschreibungsmittel für die Entwicklung von Systemanforderungsspezifikationen in der Eisenbahnsicherungstechnik zur Verfügung zu stellen. Diese Beschreibungsmittel sollen die Entwicklung einer präzisen, eindeutigen und nachvollziehbaren Systemspezifikation ermöglichen, die anschließend formal verifiziert werden kann. Durch diese formale Verifikation können die

Korrektheit und die Sicherheitskonformität der Systemanforderungsspezifikation gezeigt werden.

2 Formal verifizierbare objektorientierte Systemspezifikationen mit UML für Eisenbahnsicherungssysteme

In der Eisenbahnsicherungstechnik ist die Systemanforderungsspezifikation die wichtigste Referenz in allen Entwicklungsphasen eines Systems. Die Systemanforderungsspezifikation ist die Grundlage für den Systementwurf, die Projektierung und Implementierung des Systems und für die Durchführung der in jedem Entwicklungsschritt notwendigen Sicherheitsanalyse. Die Einhaltung der Systemanforderungen wird in jedem Entwicklungsschritt auf Basis der Beschreibungen in der Systemanforderungsspezifikation überprüft und inspiziert.

Die Entwicklung der Systemanforderungsspezifikation erfolgt durch den Auftraggeber bzw. den Systembetreiber, der die Anforderungen an das neue System mit Hilfe von Fachexperten aus der Eisenbahnsicherungstechnik niederschreibt.

2.1 Stand der Technik und Forschung

Heute werden in der Eisenbahnsicherungstechnik die Systemanforderungsspezifikationen (Lastenhefte) in natürlicher Sprache verfasst. Die Lastenhefte werden strukturiert

und sorgfältig erstellt. Die nicht funktionalen Anforderungen, wie z. B. die zu berücksichtigenden Rahmenbedingungen für die Beschaffenheit der Systemkomponenten, die Abschirmung der Stromversorgung und die Bedingungen für die Systemumgebung, werden in gesonderten Kapiteln behandelt. Weiter werden die einzuhaltenden Bestimmungen aus den Normen und Vorschriften für den Aufbau und den Betrieb des Systems in entsprechenden Abschnitten ausgeführt.

Bei der Darstellung der funktionalen Anforderungen werden zur Erläuterung einzelner Sachverhalte, wie z. B. Zugehörigkeiten von Systemkomponenten zueinander, Weg-Geschwindigkeitsverhalten und Bremskurven der Fahrzeuge etc. noch Tabellen und Grafiken verwendet, um die Eindeutigkeit und die Verständlichkeit der Beschreibungen zu erhöhen. Die Beschreibung der Systemstruktur und der Systemdynamik erfolgt jedoch hauptsächlich in natürlich sprachlicher Form. Das hat zur Folge, dass eine systematische Überprüfung der Korrektheit und der Widerspruchfreiheit der Beschreibungen äußerst schwierig und aufwändig ist.

Damit die Lastenhefte modular aufgebaut sind, erfolgt die Spezifikation jeder Hauptkomponente, jeder Komponente mit besonderer Aufgabe im System bzw. jeder wichtigen Funktionalität in einem separaten Lastenheft. So besteht z. B. die Systemanforderungsspezifikation für den Funkfahrbetrieb [ADt97] aus den Lastenheften für das Gesamtsystem, den Betrieb, den Fahrdienstleiter-Arbeitsplatz, den Fahrwegrechner, den Streckenatlas-Arbeitsplatz, die Fahrzeuge, die Fahrwegelemente, die FFB-Zentrale, die Datenkommunikation und das Bedienkonzept Fahrzeuganzeige- und Bediengerät.

Trotz dieser Aufteilung und trotz des strukturierten Aufbaus der Lastenhefte sind Anforderungen an eine Systemkomponente nicht nur in dem für diese Komponente vorgesehenen Lastenheft, sondern auch in Lastenheften für andere Systemkomponenten zu finden, die mit der Komponente an verschiedenen Prozessen beteiligt sind. Diese Eigenschaft führt zu einer mehrfachen Beschreibung von Systemprozessen, die ihrerseits zu einer Inkonsistenz der Darstellungen führen kann und die Durchführung eines Korrektheits- und Konsistenznachweises erschwert.

Dieser Problematik kann durch den Einsatz von formalen Methoden entgegengewirkt werden. Damit werden die Präzision und die Eindeutigkeit in allen Entwicklungsschritten gewährleistet und die Durchführung des Sicherheitsnachweises unterstützt.

Der Einsatz von formalen Methoden in ingenieurwissenschaftlichen und industriellen Anwendungen stößt jedoch auf diverse Hindernisse. Bereits in den 90er Jahren hat es Versuche wie z.B. [CGM96] gegeben, in denen die Sicherungssysteme im Eisenbahnwesen mit formalen Methoden spezifiziert und verifiziert wurden. Diese Versuche zeigen eine große Präzision bei der Systembeschreibung, auf deren Basis eine formale Verifikation zum Ausschluss vieler möglicher Fehler ausgeführt wird. Jedoch selbst für kleine Aufgaben werden formal beschriebene Systemspezifikationen sehr umfangreich, so dass deren Einsatz nicht mit einem vertretbaren Aufwand möglich ist. Ein weiterer Grund für einen nicht nennenswerten Einsatz von formalen Methoden in der Eisenbahn-

sicherungstechnik ist die Tatsache, dass die Ergebnisse der bereits verwendeten Methoden trotz der Erhöhung der Präzision, für die Eisenbahningenieure im Betrieb, in der Entwicklung und in der Aufsicht nicht unmittelbar verständlich, übersichtlich und handhabbar sind. Die Lücke zwischen Effizienz und Anwendbarkeit von formalen Methoden ist recht groß.

Durch den Einsatz von formal fundierten Beschreibungsmitteln wie State Charts [Har87] wurde versucht [Koc97], die Durchführung des Sicherheitsnachweises rechnergestützt zu gestalten. In [DDK99] wird auf Basis einer mit State Charts spezifizierten Beschreibung eines Bahnübergangs eine formale Verifikation der Spezifikation gegenüber funktionalen Anforderungen präsentiert. [HP03, HP03a, HP02, BHP00] stellen eine automatische Verifikation, Validation und einen Test für Eisenbahnsicherungssysteme vor, die auf einer domänenspezifischen Systembeschreibungssprache basieren. Über diese Sprache lassen sich ausführbare Implementierung, Testfälle und eine formale Verifikation generieren. In [Bjø03, Bjø03a] wird eine domänenspezifische Theorie diskutiert, die sich auf die Entwicklung der Software im Eisenbahnwesen bezieht. In den genannten Werken stehen die Aspekte der leichten Erlernbarkeit, der Anwendbarkeit und Nachvollziehbarkeit durch Eisenbahningenieure nicht im Vordergrund.

Ebenso Petri-Netze [Pet62, Pet81], Z [Spi92] und VDM [Jon90, Daw91] sind formale Methoden, die für die Spezifikation der technischen Anlagen verwendet werden. Bei Petri-Netzen steht die Spezifikation des dynamischen Verhaltens im Vordergrund. Mit diesen Beschreibungsmitteln kann die Systemstruktur nicht spezifiziert werden. Z verwendet die konstruktive Mengenlehre zur Beschreibung von Daten und Systemzuständen und Prädikatenlogik zur Darstellung der Beziehungen zwischen Ein- und Ausgabedatenmengen von Operationen. Hierfür verwendet VDM Bereichstheorie und Prädikatenlogik. Für Z und VDM sind Z++ [Lan91], Object-Z [DK+91] und VDM++ [Dür92] Erweiterungen mit objektorientierten Aspekten. Die Anwendung dieser Methoden setzt Erfahrungen im Umgang mit den genannten mathematischen Mitteln voraus.

Eine Reihe von Spezifikationstechniken basiert auf grafischen Beschreibungsmitteln. Dazu gehören SADT (Structured Analysis and Design Technique), OOA (Object Oriented Analysis) und OMT (Object Modeling Technique). SADT wurde Mitte der 70er Jahre von der Firma SofTech entwickelt und umfasst die grafische Sprache SA (Structured Analysis) für die Zusammensetzung, Strukturierung und Kommunikation von Entwurfsentitäten und die Methodik DT (Design Technique). OOA und OMT basieren auf dem objektorientierten Ansatz und stellen zur Darstellung der Objekte und ihrer Beziehungen grafische Notationen zur Verfügung. Diese Beschreibungsmittel sind anwenderfreundlich und wegen ihrer grafischen Gestalt leicht erlernbar. Trotz dieser Eigenschaft sind sie nicht für die Spezifikation und Verifikation von Sicherungssystemen geeignet, denn wegen fehlender formaler Semantik können die erstellten grafischen Spezifikationen nicht in eine formale Verifikation eingebunden werden.

2.2 Konzept und Ziel der vorliegenden Arbeit

Mit dem Ziel, für die Entwicklung und Verifikation der Systemanforderungsspezifikation adäquate Beschreibungsmittel bereitzustellen, werden in der vorliegenden Arbeit grafische Beschreibungsmittel zur Darstellung der Systemstruktur und Systemdynamik mit eindeutiger und mathematischer Semantik fundiert. Somit werden die Vorteile der grafischen Beschreibungsmittel, wie Anwenderfreundlichkeit, leichte Erlernbarkeit und einfache Nachvollziehbarkeit mit Vorteilen der formalen Methoden, wie Präzision und formaler Überprüfbarkeit vereint.

In der vorliegenden Arbeit erfolgt die Entwicklung einer Systemanforderungsspezifikation auf Basis des objektorientierten Paradigmas. In der Eisenbahnsicherungstechnik kommen Komponenten vor, die gleiche Eigenschaften und identische Funktionalitäten haben und mehrfach in dem System existieren. Eisenbahnfahrzeuge, Bahnübergänge, Weichen etc. sind Beispiele für derartige Systemkomponenten. Durch das Konzept von Objekten und Klassen bietet der objektorientierte Ansatz die Möglichkeit, die Gemeinsamkeiten solcher Systemkomponenten zu beschreiben. Die Klasse Bahnübergang beschreibt somit alle gemeinsamen statischen, strukturellen und dynamischen Eigenschaften aller Bahnübergänge vom gleichen Typ.

Ein weiteres wesentliches Charakteristikum der Eisenbahnsysteme ist es, dass manche Komponenten als Erweiterung bzw. Spezialisierung anderer Komponenten aufgefasst werden können. Die unterschiedlichen Eisenbahnfahrzeuge von verschiedenen Typen, eingleisige und mehrgleisige Bahnübergänge, verschiedene Weichentypen sind Beispiele für diese Fälle. Bei der Spezifikation dieser Komponenten können in der Objektorientierung unter Anwendung des Konzepts Vererbung die bereits spezifizierten Eigenschaften in einer Klasse von der anderen Klasse geerbt und eventuell erweitert werden. Diese Möglichkeit vereinfacht einerseits die Beschreibung von Systemkomponenten und andererseits steigert sie die Präzision der Systemanforderungsspezifikation.

Diese Eigenschaften belegen die besondere Eignung der objektorientierten Modellierung für die Spezifikation von Eisenbahnsystemen. Dabei verschafft die Beschreibung der Systemkomponenten als Objekte die Modularität und die Kompaktheit der Systembeschreibung und eine vereinfachte Handhabung bei Änderungen.

Für die Beschreibung der objektorientierten Systemanforderungsspezifikation wird hier ein objektorientiertes Modell entwickelt. Dieses Modell wird durch eine Auswahl von Beschreibungsmitteln der Unified Modeling Language (UML) dargestellt.

Die Beschreibungsmittel, die die UML zur Verfügung stellt und die bereits in verschiedenen Tools implementiert worden sind, können von der Analyse über den Entwurf bis hin zur Implementierung verwendet werden. Dadurch ist die Durchgängigkeit der Notation, die zur Formulierung der Systemanforderungsspezifikation verwendet wird, für die weiteren Entwicklungsschritte gewährleistet und eine aufwändige Neuschreibung von Informationen in jeder Entwicklungsphase nicht notwendig.

Damit wird die Eignung der angewendeten grafischen Beschreibungsmittel der UML gezeigt. Diesen Beschreibungsmitteln fehlt jedoch eine eindeutige und formale Semantik. Diese Eigenschaft erschwert einerseits eine genügend präzise Beschreibung der Sachverhalte des Anwendungsbereichs und andererseits die Durchführung einer formalen Verifikation. Um dieses Problem zu lösen, befassen sich in der letzten Zeit viele Forschungsarbeiten [UML98-04] mit einer Formalisierung der Beschreibungsmittel der UML. Diese Arbeiten basieren auf den Charakteristika der unterschiedlichen Anwendungsbereiche.

In der vorliegenden Arbeit werden für die in der Modellierung verwendeten Diagrammarten und die darin enthaltenen Notationen formale Definitionen eingeführt. Diese Definitionen sind einerseits formal und berücksichtigen andererseits die Charakteristika der Eisenbahnsicherungssysteme. Daher führen sie in erster Linie zu einer präzisen Darstellung der Systemkomponenten, ihres Verhaltens und ihrer Beziehungen zueinander. Weiter lassen sich auf Basis der eingeführten Definitionen Konsistenzbedingungen zwischen den verschiedenen Diagrammarten definieren, die unterschiedliche Sichten eines Systems darstellen.

Bei der Definition der Konzepte des objektorientierten Ansatzes wird das dynamische Verhalten der Systemkomponenten in den Vordergrund gestellt. Somit wird im Unterschied zu anderen Arbeiten z. B. für die Eigenschaften der Vererbung nicht nur die Vererbung der Attribute und Aktionen, sondern auch die Vererbung der Zustände und Transitionen vorausgesetzt.

Für eine formale Verifikation des objektorientierten Modells wird hier die Technik des Model-Checking angewendet. Model-Checking ist die Untersuchung eines Modells nach Einhaltung einer Anforderung. Im Gegensatz zum Testen, in dem unter bestimmten Annahmen lediglich ein beschränkter Teil des Modells untersucht wird, wird beim Model-Checking die Korrektheit von Anforderungen in allen möglichen Kombinationen von Zuständen überprüft, die alle Systemkomponenten in ihrem Lebenszyklus einnehmen können. Liefert das Model-Ckecking in Bezug auf eine bestimmte Anforderung ein fehlerfreies Ergebnis, besteht dann die Gewissheit, dass das Modell unter allen Umständen die Anforderung erfüllt. Im Fall eines fehlerhaften Ergebnisses, wird ein Hinweis generiert, der zur Korrektur des Modells eingesetzt werden kann. Damit kann durch Model-Checking mehr Vertrauen in das Modell gewonnen werden, als durch mehrfaches und aufwändiges Testen.

Für die Durchführung des Model-Checking ist es erforderlich, dass das zu überprüfende Modell in einer formal verifizierbaren Form vorliegt. Ein objektorientiertes Modell in der Notation der UML kann jedoch wegen fehlender formaler Semantik nicht formal verifiziert werden. In der vorliegenden Arbeit werden auf Basis der eingeführten Definitionen für die verwendeten Notationen Tranformationsschemata zur Umwandlung des objektorientierten Modells in eine formal verifizierbare Form eingeführt. Dadurch lässt sich dann das objektorientierte Modell nach Konsistenz und Sicherheitskonformität formal verifizieren.

Dieses Konzept setzt auf die Beschreibungsmittel der UML, die aufgrund ihrer grafischen Gestalt eine wachsende Popularität in der industriellen und ingenieurwissenschaftlichen Welt erfahren. Durch die in dieser Arbeit präsentierten Definitionen für die verwendeten Beschreibungsmittel der UML steigt die Präzision der Systemanforderungsspezifikationen und die Forderung der CENELEC-Normen auf eine formale Verifikation wird erfüllt.

2.3 Inhalt der weiteren Kapitel

In Kapitel drei wird zuerst die UML vorgestellt. Danach wird das Anwendungsbeispiel Funkfahrbetrieb und Auszüge eines objektorientierten Modells zur Spezifikation des eingleisigen Bahnübergangs im Funkfahrbetrieb präsentiert. Anhand dieser Auszüge wird die Darstellung der Systemstruktur und der darin enthaltenen Beziehungen zwischen den Systemkomponenten gezeigt. Weiter werden die Modellierung des Verhaltens einzelner Systemkomponenten unter der Anwendung von Zustandsdiagrammen der UML und die Beschreibung der Systemprozesse bzw. der Interaktionen unter den Systemkomponenten mit Hilfe von Sequenzdiagrammen der UML dargestellt.

In Kapitel vier erfolgt eine formale Definition einer Systemanforderungsspezifikation. Dabei werden formale Definitionen für Objekte und Klassen als Grundbegriffe eines Objektsystems eingeführt. Die Elemente zur Beschreibung von Objekten, wie z. B. Attribute, Aktionen, Ereignisse etc. werden ebenso in diesem Kapitel behandelt. Das Ziel dieses Kapitels ist, für die Grundbegriffe eines Objektssystems, die in weiteren Kapiteln der Arbeit angewendet werden, Bezeichnungen und Notationen einzuführen.

Kapitel fünf befasst sich mit der Beschreibung der Dynamik von Objekten. In diesem Kapitel werden die Ausführung von Aktionen und Aktionenfolgen, Zustände, sequenzielle, parallele und hierarchische Zustände und weitere Notationen in einem Zustandsdiagramm formal definiert. Weiter werden hier Regeln zu einer effizienten Anwendung von Zustandsdiagrammen der UML für die Beschreibung des lokalen Verhaltens der Systemkomponenten eingeführt. Zum Schluss dieses Kapitels werden formale Definitionen für die Zustandsdiagramme und die Sequenzdiagramme der UML gebracht.

In Kapitel sechs steht die formale Definition der Elemente zur Beschreibung der Systemstruktur im Mittelpunkt. Hier wird zuerst der Begriff Relation erläutert. Danach werden die Relationen Assoziation, Aggregation, Komposition und Vererbung, die in einem Klassendiagramm der UML zur Beschreibung der Beziehungen zwischen den Klassen verwendet werden, formal definiert. Diese Definitionen basieren auf den dynamischen Eigenschaften von Systemkomponenten, die in Kapitel fünf behandelt wurden. Auf Basis dieser Definitionen lassen sich Konsistenzbedingungen unter verschiedenen Spezifikationsteilen definieren, die zum Schluss dieses Kapitels erläutert werden.

Kapitel sieben hat die formale Verifikation zum Thema. In diesem Kapitel werden die Technik des Model-Checking und der Model-Checker Symbolic Model Verifier (SMV) vorgestellt. SMV kann unter anderem die in Computation Tree Logic (CTL) formulierten Prüfbedingungen in einer Kripke-Struktur überprüfen. In diesem Kapitel werden die Syntax und die Semantik von CTL sowie die Kripke-Strukturen erläutert. Damit das objektorientierte Modell von SMV überprüft werden kann, müssen die Zustandsdiagramme des Modells in Kripke-Strukturen umgewandelt werden. Der Schwerpunkt des Kapitels sieben liegt bei der Transformation von Zustandsdiagrammen der UML in die Kripke-Strukturen. In diesem Kapitel werden Schemata für die Transformation von verschiedenen Notationen eines Zustandsdiagramms in eine geeignete Kripke-Struktur vorgestellt. Am Ende dieses Kapitels werden die Grenzen des Model-Checking und die Lösungsansätze erläutert.

In Kapitel acht wird unter Anwendung der eingeführten Definitionen und Anwendungsregeln von Kapitel fünf sowie unter Anwendung der Transformationsschemata von Kapitel sieben einige Modellierungsfehler und deren Aufdeckung durch die formale Verifikation demonstriert.

In dem abschließenden Kapitel neun werden eine Zusammenfassung der Arbeit und ein Ausblick auf die Arbeiten gebracht, die auf den Ergebnissen dieser Arbeit aufbauen können.

3 Erstellung einer Systemspezifikation mit UML

In diesem Kapitel wird die Erstellung einer Systemanforderungsdefinition unter Anwendung der Beschreibungsmittel der Unified Modeling Language (UML) präsentiert.

Nach einer kurzen Einführung in die UML wird ein Ausschnitt des Betriebsverfahrens Funkfahrbetrieb (FFB) präsentiert, der hier als Anwendungsbeispiel für die Entwicklung und die Erprobung der neuen Spezifikations- und Verifikationsmethode dient. Im Anschluss daran werden die Systemeigenschaften und einige Sicherheitsanforderungen gebracht, die die Systemdefinition bereits in der ersten Definitionsphase erfüllen muss. Danach wird eine objektorientierte Modellierung mit ausgewählten Beschreibungsmitteln der UML zur Beschreibung des eingleisigen Bahnübergangs im FFB dargestellt.

3.1 Unified Modeling Language (UML)

Um eine einheitliche Spezifikationssprache für die objektorientierten Softwareentwicklungen zu definieren, wurde 1997 die Unified Modeling Language (UML) von James Rumbaugh, Grady Booch und Ivar Jacobson vorgestellt. Seitdem wird UML von der Object Management Group (OMG) weiter entwickelt und an die Anliegen der Softwareentwicklung angepasst.

Die Notationen von UML sind das Ergebnis einer mehrjährigen Zusammenarbeit zahlreicher Wissenschaftler und Vertreter der Softwareindustrie. UML ist eine Spezifikationssprache, in der verschiedene Ansätze und Darstellungsformen zur objektorientierten Modellierung zusammengeführt worden sind. Die Notationen der UML haben bereits eine weite Verbreitung in den ingenieurwissenschaftlichen und industriellen Anwendungen gefunden. Diese Beschreibungsmittel sind wegen ihrer grafischen Gestalt für die Anwender leicht erlernbar und verständlich. UML ermöglicht eine übersichtliche und leicht nachvollziehbare Darstellung der Systembeschreibung. Diese Eigenschaft ist der Hauptgrund für eine ständig wachsende Popularität der UML in den ingenieurwissenschaftlichen Anwendungen.

UML basiert auf der objektorientierten Modellierung. Dabei wird ein System als eine Menge interagierender Objekte betrachtet, deren Aktivitäten und Interaktionen, deren struktureller Aufbau und Implikationen ihrer Komponenten und Funktionalitäten zu spezifizieren ist.

UML ist eine Spezifikationssprache zur Visualisierung, Spezifikation, Konstruktion und Dokumentation von Systemen mit einem großen Anteil an Software. Sie bietet einen standardisierten Weg, verschiedene konzeptuelle und dynamische Aspekte eines Systems in unterschiedlichen Detaillierungsgraden zu beschreiben. Mit den grafischen Beschreibungsmitteln der UML können verschiedene Sichten eines Systems spezifiziert werden. In der bis jetzt von OMG endgültig verabschiedeten Version 1.5 der UML [OMG03] werden für die Beschreibung des strukturellen Aufbaus eines Systems die Paketdiagramme (packages), die Klassendiagramme, die Objektdiagramme und für die Beschreibung der dynamischen Aspekte eines Systems die Anwendungsfalldiagramme (Use Cases), Sequenzdiagramme, Kollaborationsdiagramme, Zustandsdiagramme und Aktivitätsdiagramme zur Verfügung gestellt. In der Version 2.0 der UML [OMG04], die noch nicht abgeschlossen ist, werden außer Anpassungen und Erweiterungen der genannten Diagramme noch die Übersichtsdiagramme, die Kommunikationsdiagramme, die Zeitdiagramme und die Zeitdiagramme mit Wert-Zeitachsen [Sco04, BHK04] diskutiert.

Außer den grafischen Teilen enthält UML noch die Object Constraint Language (OCL). OCL ist eine textuelle Logiksprache, die ursprünglich bei IBM (1995) entwickelt wurde. Sie wurde von UML übernommen und wird für die Beschreibung von Bedingungen für die Attributwerte, Aktionen, etc. verwendet. Mit Hilfe von OCL lassen sich Ausdrücke bezüglich der Eintragungen in einem Diagramm oder der Beziehungen zwischen den Diagrammen formulieren. Durch die Erweiterungen, die in [FM01, GZ02] vorgeschlagen werden, ist in OCL auch die Formulierung von temporallogischen Ausdrücken möglich. Da die Auswertung der OCL-Formeln weder in einem UML-Werkzeug noch von einem automatischen Verifikationssystem unterstützt wird, wird in der vorliegenden Arbeit nicht weiter auf diese Teilsprache der UML eingegangen.

Für eine objektorientierte Systemanforderungsspezifikation des Anwendungsbeispiels wird hier eine Auswahl von Diagrammen der UML verwendet. Dabei werden für die

Darstellung der Systemstruktur die Klassendiagramme und für die Beschreibung der Systemdynamik die Zustands und Sequenzdiagramme der UML angewendet. Diese Diagramme und die darin enthaltenen Notationen werden bei ihrer Anwendung erläutert. Für die weiteren Diagrammarten der UML sei auf [OMG03, OMG04, Sco04, BHK04] verwiesen.

Bevor die Systembeschreibung in der Notation der UML präsentiert wird, wird zunächst eine kurze Beschreibung des Anwendungsbeispiels Funkfahrbetrieb vorgestellt, um einen Überblick über die Anforderungen an die Systemstruktur und an das Systemverhalten zu geben.

3.2 Anwendungsbeispiel

Funkfahrbetrieb (FFB) ist ein Eisenbahnbetriebsverfahren, in dem die Betriebskomponenten FFB-Zentrale, Fahrzeug und die zustandsvariablen Fahrwegelemente per Funk miteinander kommunizieren. Die Aufgaben der Fahrwegsicherung und -steuerung, Geschwindigkeitsüberwachung und Disposition werden im FFB dezentral durch die genannten Betriebskomponenten erfüllt. Über dieses Betriebsverfahren wurde in einer Reihe von Fachbeiträgen wie [Klo96, GL97, RE97, Arm99, Hen99, Hen98] berichtet. Eine sichere Kommunikation unter den Betriebskomponenten wird durch GSM-R als Kommunikationsmedium hergestellt, das z. B. in [AC00, Ant99, ES99, Kne99, Las98, Wit99] diskutiert wird.

In [ADt97] wird FFB als ein prinzipiell neues Betriebsverfahren definiert, welches das gesamte Spektrum der Leistungsanforderungen im Bahnbetrieb abdeckt. Dabei ist FFB ein Betriebsverfahren mit minimaler ortsfester Infrastruktur für Steuerungs- und Sicherungsanlagen, das flexibel an die unterschiedlichen Leistungsanforderungen der Strecken anpassbar ist.

Die grundsätzliche Funktionsweise dieser Betriebsart wird mit der Fahrzeugsteuerung und -sicherung, der Fahrwegsteuerung und -sicherung sowie der Disposition festgelegt. Die Aufgabe der Fahrzeugsteuerung und -sicherung wird realisiert, indem die Züge sich selbst mit Hilfe von Streckenatlas, Odometer und Balisen [Hor98] orten, ihre Zugvollständigkeit prüfen und ihren Standort über Funk an die FFB-Zentrale melden. Die Dispositionsaufgaben werden von der FFB-Zentrale übernommen. Dabei verwaltet die FFB-Zentrale die Zugstandorte und weist Fahrzeugen über Funk Fahrwegabschnitte und Rangierbereiche zu. Das Prinzip der Zugfolgesicherung im FFB wird in [Pac04] erläutert. Die Überwachung der Steuerung und der Sicherung der zustandsvariablen Fahrwegelemente wird von diesen Elementen selbst durchgeführt, indem sie sich auf Anforderung der Fahrzeuge oder der FFB-Zentrale stellen, sichern und ihren Status zurückmelden.

3.2.1 Systemaufbau

Die Hauptkomponenten im FFB sind die FFB-Zentrale, die Fahrzeuge und die zustandsvariablen Fahrwegelemente. In [ADt97] sind diese Komponenten, wie folgt, beschrieben:

Die Zentrale besteht aus einem sicheren Fahrwegrechner mit Anzeige und Bedienoberfläche und ist mit einem FFB-Fahrdienstleiter besetzt. Der Fahrwegrechner dient der Zugfolgesicherung. Er überwacht Zugstandorte und weist Fahrwegabschnitte zu. Der Fahrwegrechner dient nicht zur Steuerung der Fahrwegelemente. Der FFB-Fahrdienstleiter hat im Regelbetrieb nur dispositive Aufgaben.

Die Fahrzeugausrüstung besteht aus Fahrzeugrechner mit der Aufgabe der Standortbestimmung von Zugspitze und Zugschluss sowie der Überwachung der zulässigen Geschwindigkeit, Führerraumanzeigen und Bedieneinrichtungen für Triebfahrzeugführer und Peripheriegeräten für Weg- und Geschwindigkeitsmessung und Zug- und Streckendatenspeicher (Streckenatlas).

Weiter wird für jedes einzelne Fahrwegelement die Erweiterung um ein Kommunikationsmodul zur Kommunikation mit dem Fahrzeug oder der FFB-Zentrale und um eine Elementansteuerung zur selbstständigen und permanenten Überwachung des Ordnungszustands aller im Fahrwegelement enthaltenen Subsysteme sowie zur Prüfung und Ausführung von Anweisungen vorausgesetzt.

In diesem Betriebsverfahren sind Weichen, Bahnübergänge und Schlüsselsperren zustandsvariable Fahrwegelemente, die mit ihrem Gefahrpunkt im Streckenatlas hinterlegt sind. Bei den Bahnübergängen sollten optimierte Abläufe verwendet werden. Um die Beeinträchtigung des Straßenverkehrs gering zu halten, ist z. B. das Einschalten der Bahnübergangssicherung so anzufordern, dass die Statusmeldung „BÜ gesichert“ noch rechtzeitig vor dem Erreichen des Bremsprofils im Fahrzeugrechner vorliegt. Dabei sind alle Verzögerungs- und Reaktionszeiten der beteiligten Systeme (Funk, Laufzeiten der Schranken etc.) zu berücksichtigen.

Die Freigabe des Bahnübergangs erfolgt, sobald der Zugschluss den Gefahrenbereich passiert hat entweder per Funk oder durch vorhandene streckenseitige Ausschaltmittel. Dabei soll die Bahnübergangssicherung vorzugsweise durch die eventuell vorhandenen Ausschaltmittel am Bahnübergang (z. B. Ausschaltkontakte, Gleisschleifen) erfolgen, um eine Verzögerung des Ausschaltvorgangs zu verhindern, die durch eine sichere Funkübertragung verursacht werden kann. In der vorliegenden Systembeschreibung wird diese Option durch die Kommunikation der Bahnübergangsansteuerung mit einer von der Realisierung unabhängigen sogenannten Ausschaltsensorik modelliert.

Weiter wird gefordert, dass eine hohe Verfügbarkeit und eine möglichst garantierte Ausfalloffenbarung gegeben sind. Auftretende Störungen sollen sofort an die FFB-Zentrale gemeldet werden. Für den Bahnübergang gilt zusätzlich, dass bei Überschreitung der maximal zulässigen Zeit (Grundstellungszeit), innerhalb der ein Zug einen ge-

sicherten Bahnübergang erreicht haben muss, eine entsprechende Meldung an die FFB-Zentrale gesendet wird.

In diesem Betriebsverfahren darf eine Zugfahrt erst dann stattfinden, wenn das Fahrzeug von der FFB-Zentrale eine Fahrerlaubnis für die zu befahrende Strecke erhält. In einer Fahrerlaubnis ist u.a. das Zugriffsrecht auf die Fahrwegelemente enthalten, die sich auf der Strecke befinden, auf die sich die Fahrerlaubnis bezieht. Bei der Annäherung an einen Bahnübergang sendet das Fahrzeug rechtzeitig einen Stellbefehl an den Bahnübergang und kündigt ihm die Zeit an, zu der es am Bahnübergang eintrifft. Der Bahnübergang startet den Sicherungsvorgang so rechtzeitig, dass zur Ankunftszeit der Bahnübergangsbereich für die Straßenverkehrsteilnehmer gesperrt ist. Für jeden Gefahrpunkt wie z. B. einen Bahnübergang berechnet das Fahrzeug entsprechend seiner aktuellen Geschwindigkeit eine Bremskurve, gemäß der bei Annährung an den Gefahrpunkt eine Bremsung gesteuert wird. Diese Bremskurve kann dann aufgehoben werden, wenn das Fahrzeug vom Fahrwegelement die Meldung über den gesicherten Zustand erhalten hat. Ist das zustandsvariable Fahrwegelement ein Bahnübergang, so sendet das Fahrzeug eine Statusabfrage an den Bahnübergang, bevor der Bremseinsatzpunkt erreicht ist. Folgt dieser Abfrage eine Meldung über den gesicherten Zustand des Bahnübergangs, so wird die Bremskurve aufgehoben und die Weiterfahrt ermöglicht. Anderenfalls kommt das Fahrzeug vor dem Gefahrpunkt zum Halt.

3.2.2 Systemeigenschaften

Bei der Erstellung der Systemanforderungsspezifikation in der ersten und nicht verfeinerten Definitionsphase handelt es sich grundsätzlich um die Darstellung der folgenden Eigenschaften:

1. Systemstruktur unter Angabe der Systemkomponenten und ihre Anordnung innerhalb des Systems,
2. Funktionales Verhalten der einzelnen Systemkomponenten und
3. Kommunikationsabläufe unter den Systemkomponenten bzw. Interaktionen innerhalb des Systems.

Weiter gehören zu einer Systemanforderungsspezifikation noch die Beschreibung der nicht funktionalen Anforderungen, wie des Einsatzbereichs für das neue System, der Angaben über Abmessungen und Materialien des Gehäuses, Abschirmungsmaßnahmen bei elektrischen Leitungen etc. sowie die Beschreibung der Gesetze und Vorschriften, deren Einhaltung bei Anfertigung und Erschaffung des Systems zu gewährleisten ist. Da diese aber keinen Einfluss auf eine sichere Betriebsabwicklung haben und bei einer formalen Verifikation des Systemverhaltens keine Rolle Spielen, werden diese hier bei der Erstellung eines Modells nicht berücksichtigt.

Stattdessen werden hier die Sicherheitsanforderungen betrachtet, die sich für die Gewährleistung eines sicheren Betriebes aus den Normen, Vorschriften und Gesetzen ergeben. Bezüglich des FFB sind die folgenden Anforderungen einige Beispiele für Sicherheitsanforderungen, die im Kontext des Verhaltens einzelner Betriebskomponenten formuliert worden sind:

1. Bei einer technischen Sicherungsanlage darf der Sicherungsvorgang nicht zu spät begonnen und nicht zu früh beendet werden.
2. Ein Zug darf nur abfahren, wenn ihm von der FFB-Zentrale ein Fahrweg zugewiesen worden ist.
3. Ein Zug darf nur an die Fahrwegelemente eine Anweisung/Anforderung bzw. Statusabfrage senden, die in dem ihm zugewiesenen Fahrweg liegen.
4. Ein Bahnübergang darf nur befahren werden, wenn der Zug die Meldung über die ordnungsgemäße Sicherung vom Bahnübergang erhalten hat.
5. Vor Erreichen des Bremswegabstands zum Bahnübergang muss der Zug die Meldung über die erfolgte Sicherung des Bahnübergangs erhalten haben, sonst muss er eine Bremsung entsprechend der zuvor berechneten Bremskurve einleiten.

Bei der Erstellung des objektorientierten Modells des FFB werden die Klassendiagramme der UML für die Darstellung der Systemstruktur, die Zustandsdiagramme der UML für die Beschreibung des dynamischen Verhaltens einzelner Systemkomponenten und die Sequenzdiagramme der UML für die Spezifikation der Interaktionen unter den Systemkomponenten verwendet.

Im Gegenteil zu der Sichtweise der UML, in der die Softwareobjekte aktiv oder passiv sein können, also mit der Berechnung einer Aufgabe beschäftigt sind oder ihre Aufgabe abgeschlossen haben und auf einen Wiederaufruf warten, wird für die Spezifikation der real existierenden Objekte in der Definitionsphase kein Unterschied zwischen aktiven und passiven Objekten gemacht, denn alle Objekte, also alle Systemkomponenten, sind immer aktiv. Weiter wird für die Beschreibung der Systemkomponenten von einer idealen Technologie ohne Implementierungseinschränkungen ausgegangen, so dass die Objekte auf alle definierten Ereignisse reagieren sollen. Damit kommt eine Warteschlange für die eintreffenden Ereignisse, wie sie in UML angenommen wird, nicht in Betracht.

Es werden absolute Gleichzeitigkeit, Parallelität und Nebenläufigkeit bei den Aktivitäten der Objekte zugelassen. Weiter wird von einer Kommunikation ohne Nachrichtenverlust ausgegangen. Dieser Fall wird unter den Aspekten der Fehlererkennung und -offenbarung unter Ausnahmefällen behandelt. Einige dieser Eigenschaften werden in [EW00] ausführlich erörtert.

3.3 Präzise Systemdefinition des Anwendungsbeispiels mit UML

Die Systemdefinition für das Anwendungsbeispiel eingleisiger Bahnübergang im Funkfahrbetrieb (FFB) basiert auf dem objektorientierten Paradigma. Dabei wird das System als eine Menge interagierender Objekte aufgefasst. Jedes Objekt wird durch einen Namen identifiziert und umfasst Attribute und Funktionen, die hier Aktionen genannt werden. Die Daten eines Objekts werden durch die Attributwerte bestimmt und die Dynamik der Objekte besteht aus der Änderung der Attributwerte. Die Änderung der Attributwerte erfolgt ihrerseits durch die Ausführung von Aktionen.

Für die Spezifikation des Anwendungsbeispiels wird ein objektorientiertes Modell erstellt, das die Systemstruktur und das dynamische Verhalten des Systems abbildet. Dabei werden die Systemkomponenten als interagierende Objekte betrachtet. Die Systemstruktur wird durch die Relationen zwischen den Objekten definiert und das dynamische Systemverhalten wird durch das lokale Verhalten einzelner Objekte und durch die Interaktionen unter den Objekten beschrieben.

Die Erstellung des objektorientierten Modells erfolgt schrittweise mit dem Ziel, die Systemeigenschaften und ihre Implikationen zu erkennen und zu spezifizieren. Dabei werden die einzelnen Systemkomponenten identifiziert und ihre Aufgaben bestimmt, die sie zur Realisierung der Funktionalitäten des Systems eigenständig oder kollektiv durchführen. In den nächsten Abschnitten wird die Erstellung des Modells erläutert. Dabei wird auch diskutiert, wie die Anforderungen im Anwendungsbeispiel in dem Modell abgebildet werden.

3.3.1 Objektorientiertes Modell des FFB

In dem objektorientierten Modell wird der Begriff *Objekt* für eine reale bzw. konzeptuelle Einheit verwendet. Damit sind eine bestimmte Bahnübergangssicherungsanlage, eine Lichtzeichenanlage, ein Fahrweg, der aus Fahrwegelementen, Balisen und Gleisabschnitten besteht, Beispiele für Objekte. Der Begriff *Klasse* umfasst dann alle Objekte mit dem gleichen Aufbau und mit der identischen Ausstattung. Die Klasse *Lichtzeichenanlage* beschreibt alle Lichtzeichenanlagen vom gleichen Typ und die Klasse *Fahrweg* beschreibt alle Fahrwege auch, wenn sich diese in der Anzahl ihrer Komponenten unterscheiden.

Die Kommunikation unter den Objekten der Klassen ist ereignisgesteuert und erfolgt durch Aufrufe der entsprechenden Aktionen in den jeweiligen Objekten. Ein Ereignis, das in einem Objekt eine Reaktion aufruft, ist selbst das Ergebnis einer gleichnamigen Aktion eines anderen Objekts. Daher werden hier die Ereignisse mit dem Namen der Aktionen benannt, die sie veranlassen. Die Zeit und die Überwachung der Aktionen und

Ereignisse, die Zeit verbrauchen, wurde in diesem Modell explizit durch Kommunikation mit dem Objekt Timer modelliert.

In diesem Modell werden die Anordnung der Objekte in dem gesamten System und die Beziehungen unter den Objekten Systemstruktur genannt. Die Systemstruktur wird durch Klassendiagramme präsentiert, in denen die Klassen und die Beziehungen unter den Klassen abgebildet werden. Diese Beziehungen werden durch die Relationen Assoziation, Aggregation, Komposition und Vererbung spezifiziert.

Um die Dynamik der einzelnen Systemkomponenten darzustellen, wird für jede Klasse ein Zustandsdiagramm entworfen, welches das lokale Verhalten der Objekte dieser Klasse beschreibt. Dieses Verhalten ist die Aktivitäten von Objekten der Klasse in ihrem gesamten Lebenszyklus und besteht aus der Ausführung von Aktionen, Einnahme von Zuständen und Zustandswechseln, die in Abhängigkeit von Ereignissen, Bedingungen bzw. weiteren Aktionsausführungen stattfinden. Dieses Verhalten umfasst sowohl den regulären und fehlerfreien als auch den fehlerhaften Fall.

Die Interaktionen unter den Objekten werden durch Sequenzdiagramme skizziert. Dabei werden jeweils ein Sequenzdiagramm für den fehlerfreien Fall und mehrere Sequenzdiagramme für die fehlerhaften Fälle entworfen. Da diese Abläufe als Systemanforderungen gelten, sind sie zum Teil Sicherheitsanforderungen, die als Korrektheitsbedingungen in Form von Spezifikationen bei der Durchführung der zukünftigen formalen Verifikation eingesetzt werden.

3.3.2 Erstellung des objektorientierten Modells

Das objektorientierte Modell des Anwendungsbeispiels eingleisiger Bahnübergang im FFB besteht aus 17 Klassen, die nicht alle hier präsentiert werden. In den folgenden Abschnitten werden Auszüge aus diesem Modell gebracht, die die Systemstruktur und das dynamische Systemverhalten abbilden. Anhand dieser Bespiele sollen die Anwendung des objektorientierten Ansatzes, die Erläuterung der verwendeten Modellierungstechnik und die Bedeutung der angewendeten Beschreibungsmittel der UML erörtert werden.

3.3.3 Systemstruktur im FFB

Die Systemstruktur wird durch Klassendiagramme präsentiert, in denen die Klassen und die Beziehungen unter den Klassen abgebildet werden. In einem Klassendiagramm (Abb. 3.1-3.4) werden die Klassen durch Vierecke dargestellt. Jedes Viereck besteht aus drei Teilen. In dem oberen Teil steht der Klassenname, der mittlere Teil enthält die Attribute der Klasse und in dem unteren Teil werden die Aktionen aufgelistet, die die Objekte der Klasse ausführen.

In dem objektorientierten Modell des Anwendungsbeispiels sind die drei Komponenten FFB-Zentrale, Fahrzeuge und Fahrwegelemente die ersten Klassen. Die Attribute und

die Aktionen dieser Klassen lassen sich mit Hilfe von Teilaufgaben definieren, die sich aus den Hauptaufgaben Fahrwegsteuerung und -sicherung, Zuggeschwindigkeitsüberwachung und Zugfolgesicherung für jede Komponente ergeben.

Ein Fahrzeug benötigt eine Fahrerlaubnis, um eine Zugfahrt durchzuführen. In der Fahrerlaubnisanforderung stehen entsprechend den Streckendaten in dem Streckenatlas der Anfang und das Ende der zu befahrenden Strecke. Weiter soll ein Fahrzeug den Fahrwegelementen auf der zu befahrenden Strecke einen Stellbefehl senden. Daher werden die Aktionen *fahrerlaubnis_anfordern*(), *element_anfordern*() etc. für die Klasse *Fahrzeug* definiert.

Die FFB-Zentrale ist für die Erteilung einer Fahrerlaubnis und die Zuweisung eines Fahrwegs zuständig. In einer Fahrerlaubnis sind unter anderem die Zugriffsrechte auf die Fahrwegelemente enthalten, die sich auf der angeforderten Strecke befinden. Dementsprechend werden in der Klasse *FFB-Zentrale* die Aktionen *fahrerlaubnis_überprüfen*(), *fahrerlaubnis_senden*() etc. definiert.

Schließlich werden in der Klasse *Fahrwegelement* entsprechend den für die zustandsvariablen Fahrwegelemente vorgesehenen Aufgaben die Aktionen *sicherung_starten*(), *status_melden*() etc. definiert, denn jedes zustandsvariable Fahrwegelement soll sich auf Anforderung sichern und seinen Status auf Anfrage melden.

Für die Aufgaben, wie *streckenatlas_aktualisieren*(), *streckenatlas_verteilen*() etc., die keiner der genannten Klassen zuzuteilen sind, wird die Klasse *Verwaltung* eingeführt. Diese Klasse ist ebenso für die Inbetriebnahme und die Außerbetriebnahme aller Systemkomponenten zuständig. Damit ergibt sich die grobe Systemstruktur wie in Abb. 3.1.

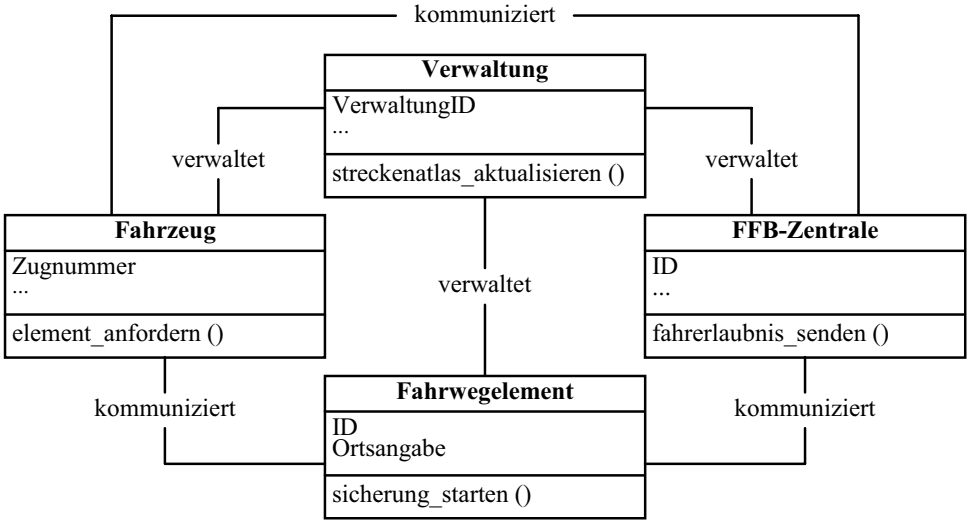


Abb. 3.1 Systemstruktur des FFB

Mit der Identifizierung der Klassen werden in der Spezifikation der Systemstruktur auch die Beziehungen zwischen Klassen spezifiziert. In UML können die Beziehungen zwischen den Klassen mit Hilfe der Relationen Assoziation, Aggregation, Komposition und Vererbung dargestellt werden. Diese Relationen werden hier für die Beschreibung der Beziehungen zwischen den Systemkomponenten in FFB verwendet.

In Abb. 3.1 zeigen die Verbindungslinien die Assoziationen unter den abgebildeten Klassen. In UML werden Assoziationen verwendet, um in einem Modell zwischen verschiedenen Entitäten eine semantische Beziehung zu definieren. Diese Entitäten können Objekte, Klassen, Attribute etc. sein. Ausgehend von den Entitäten, für die die Assoziation definiert wurde, bzw. abhängig von der Art der Beziehung zwischen den beteiligten Entitäten, haben die Assoziationen unterschiedliche Eigenschaften. Daher wird in UML für die Assoziation keine feste Definition gegeben. In der vorliegenden Arbeit bezeichnet eine Assoziation die folgende Beziehung zwischen den Klassen.

3.3.3.1 Assoziation

> Wenn die Objekte zweier Klassen miteinander kommunizieren, so existiert eine Assoziation unter diesen Klassen.
>
> Jede Assoziation zeigt, dass die Objekte der einen Klasse Aktionen ausführen, die für die Objekte der anderen Klasse als Ereignisse gelten und dementsprechend auf diese Ereignisse reagieren.

Mit einer Assoziation kann auch eine Kardinalität angegeben werden, die die Anzahl der assoziierenden Objekte jeder Klasse festlegt. In Kapitel sechs werden die Relation Assoziation und die Kardinalitätsangabe präzise beschrieben.

Der nächste Schritt bei der Entwicklung des Modells besteht darin, für die Objekte der bereits identifizierten und festgelegten Klassen die Struktur bzw. den Aufbau zu beschreiben. In diesem Schritt werden die Beziehungen zwischen den Komponenten und ihren Teilkomponenten untersucht.

In UML wird die Bezeichnung zwischen den Objekten und ihren Teilobjekten durch die Relationen Aggregation (Shareable Aggregation) und Komposition (Composite Aggregation) dargestellt. Dabei bezeichnet eine Aggregation die Zusammengehörigkeit der Objekte, in der das Teilobjekt zur gleichen Zeit bzw. zu verschiedenen Zeiten den unterschiedlichen Aggregaten gehören kann. In diesem Fall bleibt das Teilobjekt nach dem Auflösen des Aggregats in dem System erhalten. Die Komposition ist aber eine strenge Beziehung, in der das Teilobjekt in seinem Lebenszyklus ausschließlich einem Ganzen gehört und von ihm existenzabhängig ist, das heißt, das Teilobjekt wird zusammen mit dem Ganzen erzeugt und mit der Auflösung des Ganzen aus dem System gelöscht. Diese Relationen und ihre Eigenschaften werden in Kapitel sechs ausführlich diskutiert.

In [BH99] werden ausgehend von ontologischen Aspekten verschiedene Arten der Zusammengehörigkeit von Objekten zueinander erörtert. Da in der Systemanforderungsspezifikation nicht die Beschreibung aller Arten der Zusammengehörigkeit von Objekten wichtig ist, werden in der vorliegenden Arbeit die Relation Komposition für eine exklusive und existenzabhängige Zusammengehörigkeit von Objekten und die Relation Aggregation für eine allgemeine Zusammengehörigkeit der Objekte verwendet. In [Ara02] werden die Eigenschaften dieser Relationen auf Basis der dynamischen Eigenschaften der beteiligten Objekte diskutiert.

Da es sich bei der Relation Aggregation um eine allgemeine Art von Zusammengehörigkeit handelt, wird das Verhältnis unter den beteiligten Objekten in den weiteren Entwicklungsphasen spezifiziert, wenn in der Systemanforderungsspezifikation zwischen Objekten diese Relation definiert ist. In Kapitel sechs werden unter anderem die Relationen Aggregation und Komposition eingehend diskutiert. Für die Anwendung dieser Relationen genügen hier jedoch die folgenden Definitionen:

3.3.3.2 Aggregation

> Die Relation Aggregation wird verwendet, um eine physikalische oder konzeptuelle Zugehörigkeit von Objekten einer Klasse zu Objekten einer anderen Klasse zu bezeichnen. Wie bei einer Assoziation kann auch hier eine Kardinalität angegeben werden, die die Anzahl der Objekte, die an einer Zugehörigkeitsrelation beteiligt sind, festlegt.

In einem Klassendiagramm wird die Aggregation durch eine nicht gefüllte Raute bezeichnet (Abb. 3.2), die auf die Klasse des Ganzen zeigt und durch eine Linie mit der Klasse des Teilobjekts verbunden wird.

Für die Aggregation wird hier vorausgesetzt, dass diese Relation azyklisch ist, obwohl das in UML nicht gefordert wird. Da diese Relation eine physikalische bzw. konzeptuelle Zugehörigkeit von real existierenden Komponenten eines Systems bezeichnet, würde ein Zyklus bedeuten, dass das Ganze wiederum ein Teil des Teils ist.

Gibt es unter dem Ganzen und seinen Teilobjekten eine strenge Zusammengehörigkeit, so wird die Relation Komposition mit folgenden Eigenschaften verwendet:

3.3.3.3 Komposition

> Die Relation Komposition zeigt eine exklusive und existenzabhängige Beziehung zwischen dem Ganzen und seinen Teilobjekten. Dabei gibt es die folgenden Eigenschaften unter den beteiligten Objekten:
>
> 1. Unter dem Ganzen und seinen Teilen ist eine Assoziation gegeben.
> 2. Die Teilobjekte werden von dem Ganzen und von keinem anderen Objekt initialisiert und gelöscht.

3. Das Ganze wird von keinem seiner Teilobjekte erzeugt bzw. gelöscht.
4. Die Existenz von Teilobjekten ist von der des Ganzen abhängig und umgekehrt.
5. Die Komponenten gehören in ihrer ganzen Lebensdauer ausschließlich zu einem einzigen Ganzen.

In einem Klassendiagramm wird die Komposition durch eine gefüllte Raute bezeichnet (Abb. 3.3), die auf die Klasse des Ganzen zeigt und durch eine Linie mit der Klasse des Teilobjekts verbunden wird. Wie bei einer Aggregation kann auch bei einer Komposition eine Kardinalität angegeben werden, die die Anzahl der Objekte, die an einer exklusiven und existenzabhängigen Zugehörigkeitsrelation beteiligt sind, festlegt.

Aus demselben Grund wie bei der Relation Aggregation wird auch bei Komposition verlangt, dass diese Relation azyklisch ist.

Generell sind die Relationen Assoziation, Aggregation und Komposition nicht bidirektional, das heißt, wenn die Objekte einer Klasse in einer Beziehung mit den Objekten einer zweiten Klasse stehen, so steht nicht unbedingt jedes Objekt der zweiten Klasse mit Objekten der ersten Klasse in einer entsprechenden Beziehung.

Mit Hilfe der genannten Relationen können die ersten Klassen des Modells, die in Abb. 3.1 dargestellt worden sind, weiter spezifiziert werden. Zunächst werden weitere Objekte betrachtet, die sich auf einer Strecke befinden und für den Betrieb im FFB relevant sind.

Außer zustandsvariablen Fahrwegelementen befinden sich auf den Strecken noch Balisen, die dazu dienen, den Fahrzeugen absolute Ortinformationen zu liefern. Damit besteht ein Fahrweg im FFB aus zustandsvariablen Fahrwegelementen, Gleisabschnitten und Balisen. Bei der Zuweisung eines Fahrwegs an einem Fahrzeug werden ihm auch die zustandsvariablen Fahrwegelemente, Gleisabschnitte und Balisen zugewiesen, die dem Fahrweg gehören. Wenn diese Teilobjekte von dem Fahrzeug ganz oder teilweise freigegeben werden, so wird der Fahrweg ganz oder teilweise freigegeben. Nach der Freigabe der einzelnen Teilobjekte bzw. nach einer eventuellen Rückgabe des zugewiesenen Fahrweges bleiben die Teilobjekte in dem System erhalten und sie können für die Bildung weiterer möglicher Fahrwege eingesetzt werden.

Das Verhältnis zwischen einem Fahrweg und seinen Teilobjekten ist ein Beispiel für eine Zusammengehörigkeit der Objekte, in der die Objekte nicht voneinander existenzabhängig sind.
In diesem Beispiel gibt es auch keine Verhaltensabhängigkeit zwischen den genannten Objekten. Die Teilobjekte kommunizieren weder unter sich noch mit dem Fahrweg. Auch der Status des Fahrweges, der sich von dem Status seiner Teilobjekte herleiten

lässt, wird nicht über eine Kommunikation mit den Teilobjekten, sondern von der FFB-Zentrale bestimmt, der die notwendigen Informationen von dem Fahrzeug zur Verfügung gestellt werden.

Das Verhältnis zwischen der Klasse *Fahrweg* und ihren Teilen wird in Abb. 3.2 mit Hilfe der Relation Aggregation dargestellt.

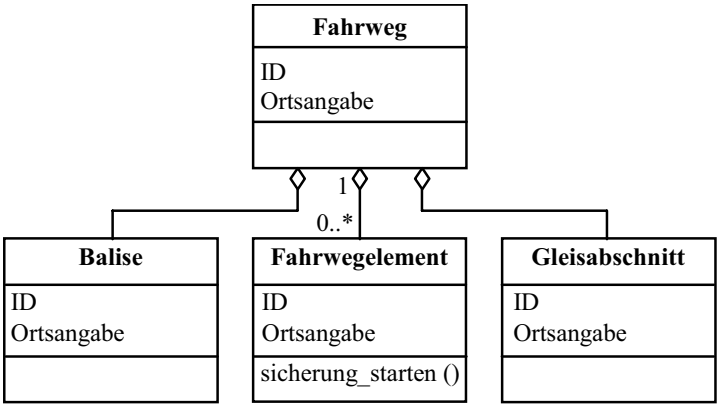


Abb. 3.2 Fahrweg und seine Komponenten

Mit der Identifizierung der Teilobjekte für die Klasse *Fahrweg* wird die Grobstruktur erweitert. Zum einem wird die Klasse *Fahrwegelement* eingeführt, die die zustandsvariablen Fahrwegelemente beschreibt. Im FFB sind Bahnübergänge, Weichen und Schlüsselsperren zustandsvariable Fahrwegelemente. Weiterhin werden die Klassen *Balise* und *Gleisabschnitt* im Modell aufgenommen. Diese drei Klassen sind dann für die Objekte der Klasse *FFB-Zentrale* bei der Überwachung der zu befahrenden Strecken und bei der Vergabe der angeforderten Fahrerlaubnisse von Bedeutung.

Mit der Beschreibung der Klasse *Fahrwegelement* wird das Modell weiter entfaltet. Jedes zustandsvariable Fahrwegelement kommuniziert per Funk mit den Fahrzeugen oder mit der FFB-Zentrale. Daher verfügen alle zustandsvariablen Fahrwegelemente über ein Kommunikationsmodul. Weiter ist die Überwachung der Sicherungsabläufe die Aufgabe der Fahrwegelemente selbst. Daher werden zwei neue Klassen *Kommunikationsmodul* und *Timer* in das Modell aufgenommen, deren Objekte u.a. Teile der Objekte der Klasse *Fahrwegelement* sind. In Abb. 3.3 wird das Verhältnis zwischen der Klasse *Fahrwegelement* und seinen Teilen mit der Relation Komposition dargestellt.

Eine weitere Relation, die in dem objektorientierten Modell des Anwendungsbeispiels verwendet wird, ist die Vererbung. In UML wird unter Vererbung eine Relation definiert, die die Übergabe aller Eigenschaften einer Klasse, wie Attribute, Aktionen etc. an eine andere Klasse umfasst. Dabei werden die Klasse, die die Eigenschaften vererbt, als *Generalisierungsklasse* und die Klasse, die die Eigenschaften erbt und eventuell diese erweitert bzw. überschreibt, als *Spezialisierungsklasse* bezeichnet.

Mit Hilfe von Vererbung werden im objektorientierten Modell des Anwendungsbeispiels z. B. die Bahnübergänge als eine Spezialisierung von zustandsvariablen Fahrwegelementen definiert. Damit erbt die Klasse *Bahnübergang*, die die eingleisigen Bahnübergänge beschreibt, alle Eigenschaften der Klasse *Fahrwegelement* und erweitert diese mit ihren Eigenen. Jedes Objekt der Klasse *Bahnübergang* hat alle Attribute und Aktionen und alle Teilobjekte, die in der Klasse *Fahrwegelement* definiert worden sind. Das bedeutet, dass jedes Objekt der Klasse *Bahnübergang* ein Objekt der Klasse *Timer* und ein Objekt der Klasse *Kommunikationsmodul*. Zusätzlich haben die Objekte der Klasse *Bahnübergang* weitere Teilobjekte der Klassen *Lichtzeichenanlage*, *Schrankenanlage* und *Ausschaltsensorik*. Abb. 3.3 zeigt die Klassen *Fahrwegelement* und *Bahnübergang*, ihre Teile und ihre Beziehungen zueinander.

Die gleiche Art Beziehung besteht auch zwischen eingleisigen und mehrgleisigen Bahnübergängen (MG-Bahnübergang), das heißt, die Klasse *MG-Bahnübergang* erbt alle Eigenschaften der Klasse *Bahnübergang* und erweitert diese mit ihren eigenen Eigenschaften, worauf hier nicht weiter eingegangen wird.

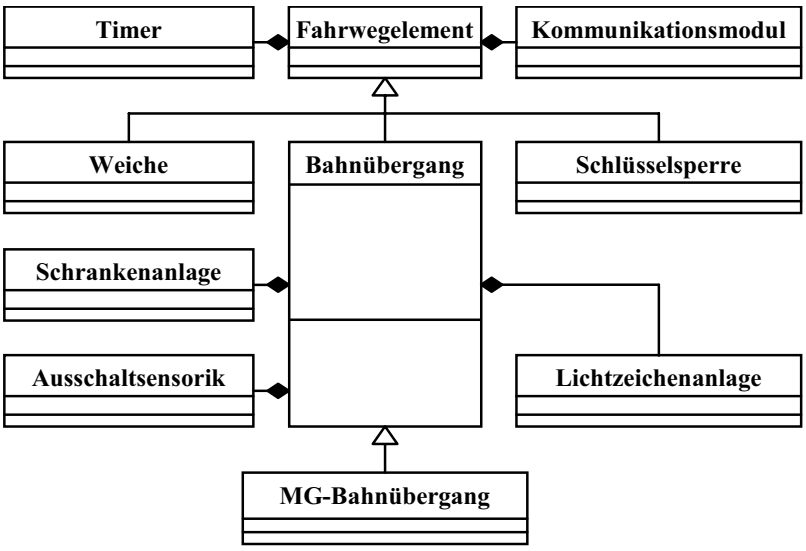


Abb. 3.3 Klassendiagramm der Klassen Fahrwegelement und Bahnübergang

Die einfache Übernahme von Aktionen der Klasse *Fahrwegelement* allein kann jedoch nicht die Ausführung dieser Aktionen durch die Objekte der Klasse *Bahnübergang* als Sicherungsanlagen in gewünschter Reihenfolge gewährleisten. Daher muss die Vererbung auch den Kontext der Ausführung von Aktionen mit beinhalten. Deshalb unterliegt die Relation Vererbung in der vorliegenden Arbeit einer strengeren Definition, als der in UML. Durch diese Definition werden nicht nur die statischen Eigenschaften von Objekten, sondern auch ihr dynamisches Verhalten, wie in nächsten Abschnitten gezeigt wird, vererbt.

3.3.3.4 Vererbung

Vererbung ist die Übergabe aller Attribute, Aktionen, Relationen und des dynamischen Verhaltens der Generalisierungsklasse an die Spezialisierungsklasse. Die Spezialisierungsklasse erweitert bzw. überschreibt eventuell die geerbten Eigenschaften.

In einem Klassendiagramm wird die Vererbung durch ein Dreieck bezeichnet, das auf die Generalisierungsklasse zeigt und durch eine Linie mit der Spezialisierungsklasse verbunden wird.

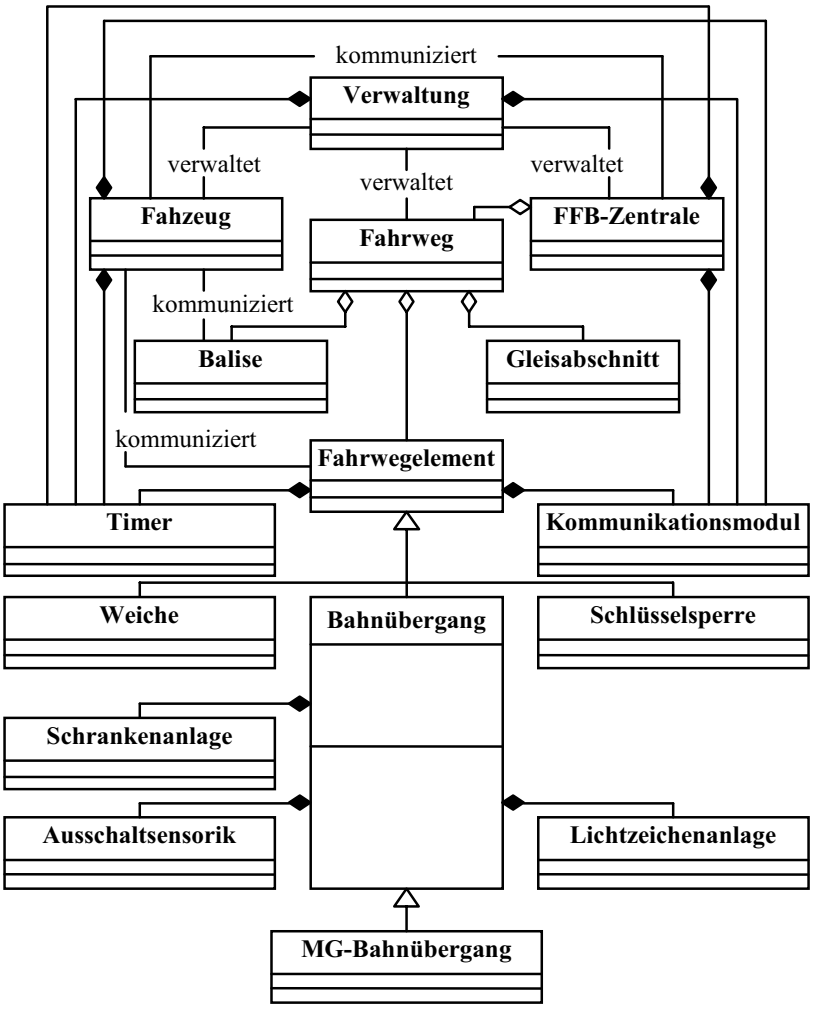


Abb. 3.4 Systemstruktur

3.3.3.5 Mehrfache Vererbung

Erbt eine Klasse die Eigenschaften mehrerer Klassen, so heißt diese Art von Vererbung mehrfache Vererbung.

Aufgrund der Vererbung des dynamischen Verhaltens, die in nächsten Abschnitten beschrieben wird, wird in der vorliegenden Arbeit keine mehrfache Vererbung verwendet.

In Kapitel sechs werden alle Relationen Assoziation, Aggregation, Komposition und Vererbung und ihre Eigenschaften präzise beschrieben.

Mit den genannten Klassen wird die Systemstruktur wie in Abb. 3.4 weiter vervollständigt. Weitere Erweiterungen des Modells sind die Teilkomponenten der Klassen *Fahrzeug*, *FFB-Zentrale* und *Verwaltung*. Die Klasse *Fahrzeug* hat die Klassen *Kommunikationsmodul*, *Timer*, *Streckenatlas*, *Odometer* und *Bremse* als Teilkomponenten. Die Klasse *FFB-Zentrale* besitzt die Teilkomponenten *Kommunikationsmodul*, *Timer*, *Streckenatlas* und *Fahrweg* und die Klasse *Verwaltung* hat die Klassen *Kommunikationsmodul*, *Timer* und *Fahrweg* als Teilkomponenten. Auf eine grafische Darstellung aller genannten Teilkomponenten wird hier verzichtet.

3.3.4 Systemdynamik im FFB

Die Dynamik des Systems besteht aus dem internen Verhalten einzelner Objekte und aus den Interaktionen zwischen den Objekten. In dem objektorientierten Modell des Anwendungsbeispiels wird das interne Verhalten der Objekte mit Hilfe der Zustandsdiagramme der UML spezifiziert. Dabei wird für jede Klasse ein Zustandsdiagramm entworfen, das das Verhalten aller Objekte dieser Klasse beschreibt. Ein Zustandsdiagramm (Abb. 3.5-3.8) besteht aus Zuständen und Zustandsübergängen. Jeder Zustand wird in Form eines gerundeten Vierecks dargestellt, in dem der Zustandsname und die Aktivitäten notiert werden, die in diesem Zustand ausgeführt werden. Weiter kann ein Zustand weitere Teilzustände haben. Werden die Aktivitäten in den Teilzuständen parallel ausgeführt, so werden diese Teilzustände durch gestrichelte Linien getrennt. Die Zustandsübergänge werden durch Transitionspfeile präsentiert, die von einem Zustand zu seinem Folgezustand gezeichnet werden. Der Start und das Ende der Aktivitäten werden in einem Zustandsdiagramm durch das Startsymbol (ein gefüllter Kreis) und den Endzustand (ein umrandeter gefüllter Kreis) gekennzeichnet.

Die Beschreibung einzelner Systemabläufe erfolgt durch Sequenzdiagramme der UML. Mit diesen Diagrammen können sowohl die gewünschten als auch die nicht gewünschten Fälle beschrieben werden. Hier wird zuerst die Beschreibung des internen Verhaltens der Hauptkomponenten anhand der Klassen *Fahrwegelement*, *Bahnübergang*, *FFB-Zentrale* und *Fahrzeug* erläutert und die Modellierungstechnik zur Erfassung der funktionalen Anforderungen gezeigt.

3.3.4.1 Internes Verhalten von Systemkomponenten im FFB

In den Lastenheften zum FFB [ADt97] werden bestimmte funktionale Anforderungen für alle Arten der zustandsvariablen Fahrwegelemente, die in Form von Bahnübergängen, Weichen oder Schlüsselsperren existieren können, definiert. Diese allgemeinen

Anforderungen motivieren das Definieren eines allgemeinen Verhaltens, das dann für jeden bestimmten Typ von Fahrwegelementen gilt.

3.3.4.1.1 Internes Verhalten von zustandsvariablen Fahrwegelementen

In [ADt97] wird für jedes Fahrwegelement verlangt, dass es selbstständig den Sicherungsvorgang startet und beendet. Jedes Fahrwegelement soll den Erhalt eines Stellbefehls quittieren und auf eine Statusabfrage mit einer Meldung über den gesicherten oder nichtgesicherten Zustand reagieren. Fahrwegelemente sollen in der Lage sein, durch das permanente Ausführen eines Tests eigene Fehler zu erkennen und diese an die FFB-Zentrale zu melden. Ein Defektes Fahrwegelement soll auf Stellbefehle bzw. auf Statusabfragen mit einer Meldung über seinen defekten Zustand reagieren. Schließlich soll zur Erhöhung der Verfügbarkeit und Aufrechterhaltung des Betriebs jedes Fahrwegelement die Möglichkeit zur manuellen Steuerung bieten, damit es vom Personal bedient werden kann, wenn z. B. das Fahrwegelement wegen des Ausfalls der Kommunikationsausrichtung defekt ist. Die genannten Anforderungen an Fahrwegelemente lassen sich zu einem allgemeinen Verhalten für alle Fahrwegelemente spezifizieren. Abb. 3.5 zeigt dieses allgemeine Verhalten. Dabei steht t_{akt} für die aktuelle Zeit.

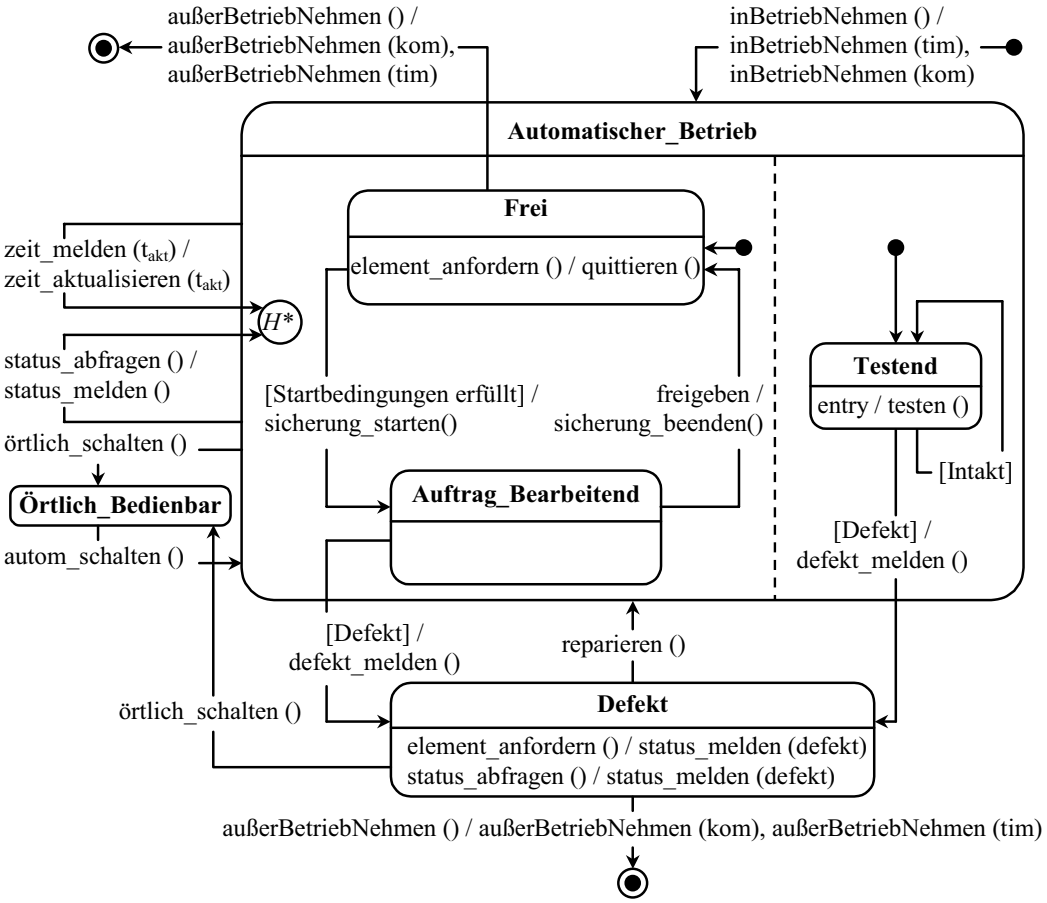


Abb. 3.5 Zustandsdiagramm der Klasse Fahrwegelement

Nach dieser Beschreibung besteht der Lebenszyklus eines Objekts der Klasse *Fahrwegelement* aus drei Zuständen *Automatischer_Betrieb*, *Örtlich_Bedienbar* und *Defekt*.

Nachdem ein Objekt dieser Klasse als Reaktion auf das Ereignis *inBetriebNehmen*() entsteht, erzeugt es zuerst mit der Ausführung der Aktionen *inBetriebNehmen*(*tim*) und *inBetriebNehmen*(*kom*) seine exklusiven und existenzabhängigen Teilobjekte aus den Klassen *Timer* und *Kommunikationsmodul*. Nach der Ausführung dieser Aktionen geht das Objekt in den Zustand *Automatischer_Betrieb* über. In diesem Zustand finden zwei verschiedene Prozesse unabhängig voneinander statt, die in zwei parallelen Regionen modelliert worden sind.

In der rechten Region befindet sich das Objekt in dem Teilzustand *Testend*, in dem die Aktion *testen*() permanent ausgeführt wird. Es handelt sich hier um Selbsttests. Diese Aktion wird von jedem speziellen Typ von zustandsvariablen Fahrwegelementen spezifisch ausgeführt. Wird nach dem Ausführen dieser Aktion festgestellt, dass beim Objekt kein Fehler vorliegt ([*Intakt*]), so wird die Aktion *testen*() erneut ausgeführt. Ansonsten ([*Defekt*]) wird der gesamte Zustand *Automatischer_Betrieb* verlassen und das Objekt geht in den Zustand *Defekt* über.

In dem Zustand *Defekt* reagiert das Objekt auf die Ereignisse *element_anfordern*() bzw. *status_abfragen*() mit der Ausführung der Aktion *status_melden*(*defekt*). Dieser Zustand wird mit einem der Ereignisse *reparieren*() bzw. *außerBetriebNehmen*() verlassen, die von dem Objekt der Klasse *Verwaltung* ausgeführt werden. In dem letzten Fall führt das Objekt noch die Aktionen *außerBetriebNehmen*(*kom*) und *außerBetriebNehmen*(*tim*) aus und entfernt damit seine Teilobjekte aus dem System.

In der linken Region des Zustands *Automatischer_Betrieb* führt das Objekt die Aufgabe der automatischen Erstellung und Aufhebung der Sicherung. In dieser Region befindet sich das Objekt zuerst in dem Teilzustand *Frei* und wartet auf Stellbefehle. Mit dem Ereignis *element_anfordern*(), das von den Objekten der Klasse *Fahrzeug* ausgelöst wird, führt das Objekt die Aktion *quittieren*() aus. Durch diese Aktion quittiert das Objekt den Erhalt des Stellbefehls. Der Teilzustand *Frei* wird verlassen, sobald [*Startbedingungen erfüllt*] gilt. Jeder Typ von Fahrwegelementen hat seine spezifischen Startbedingungen. Während ein Bahnübergang bis zum Einschaltzeitpunkt warten muss, kann eine Weiche sofort nach dem Erhalt des Stellbefehls mit der Erstellung der Sicherung beginnen, das heißt, die genannte Bedingung ist bei einem Bahnübergang frühestens zum Einschaltzeitpunkt und bei einer Weiche sofort erfüllt.

Nachdem [*Startbedingungen erfüllt*] gilt, wird der Teilzustand *Frei* verlassen. Dabei wird die Aktion *sicherung_starten*() ausgeführt und das Objekt geht in den Teilzustand *Auftrag_Bearbeitend* über. In diesem Zustand befasst sich das Objekt mit der Bearbeitung des Stellbefehls. Tritt während dieser Bearbeitung ein Fehler [*Defekt*] auf, so wird wiederum der gesamte Zustand *Automatischer_Betrieb* verlassen und das Objekt geht in den Zustand *Defekt* über. Der Teilzustand *Auftrag_Bearbeitend* wird im Regelfall mit dem Ereignis *hinausfahren*() verlassen, das von dem Objekt der Klasse *Fahrzeug* ausge-

löst wird. Nach dem Verlassen dieses Teilzustands geht das Objekt in den Teilzustand *Frei* über und wartet auf den nächsten Stellbefehl. Schließlich kann ein Objekt der Klasse *Fahrwegelement* aus dem System entfernt werden, wenn es sich in dem Teilzustand *Frei* befindet und bei ihm keine Anforderung vorliegt.

Jedes zustandsvariable Fahrwegelement soll auf eine Statusabfrage mit einer Meldung über seinen aktuellen Zustand reagieren. Diese Anforderung wurde in dem Zustand *Automatischer_Betrieb* durch eine entsprechende Transition in den Deep-History-State H^* wiedergegeben. Durch diese Modellierung bleibt das Objekt nach der Ausführung der Aktion *status_melden*() in dem Teilzustand, in dem es sich bei dem Erhalt des Ereignisses *status_abfragen*() befindet. Dasselbe gilt für die Ausführung der Aktion *zeit_aktualisieren*(t_{akt}), die auf das von dem Objekt der Klasse *Timer* ausgelöste Ereignis *zeit_melden*(t_{akt}) folgt, dadurch erhält das Objekt der Klasse *Fahrwegelement* die aktuelle Zeit.

Durch den Zustand *Örtlich_Bedienbar* ist die Möglichkeit zur manuellen Steuerung des Objekts gegeben, damit es ggf. vom Personal bedient werden kann, wenn z. B. das Fahrwegelement defekt ist. Der Übergang zu diesem Zustand ist durch das Ereignis *örtlich_schalten*() definiert, das durch die Betätigung einer entsprechenden Taste erfolgt, zu der das Personal Zugand hat. Nach der manuellen Steuerung kann das Objekt wieder in den Zustand *Automatischer_Betrieb* versetzt werden, wenn das Ereignis *autom_schalten*() von dem Personal durch die Betätigung der entsprechenden Taste ausgelöst wird.

Das hier beschriebene allgemeine Verhalten für Fahrwegelemente lässt sich unter Anwendung des Konzepts der Vererbung von den Objekten der Klasse *Bahnübergang* übernehmen und erweitern. In [AG00] wird diese Art von Vererbung ausführlich diskutiert. Dabei werden von der Spezialisierungsklasse alle Attribute, Aktionen, Zustände und Transitionen der Generalisierungsklasse sowie alle Beziehungen zu den anderen Klassen übernommen. In Kapitel sechs wird die Vererbung formal beschrieben. Im Folgenden wird die Vererbung der dynamischen Eigenschaften gezeigt.

3.3.4.1.2 Internes Verhalten des eingleisigen Bahnübergangs

Ein Bahnübergang ist ein zustandsvariables Fahrwegelement, das die Durchführung des Sicherungsvorgangs über die Steuerung einer Lichtzeichenanlage (lzA) und die einer Schrankenanlage (schA) erfüllt. Laut Definition ist ein technisch gesicherter Bahnübergang im gesicherten Zustand, wenn die Lichtzeichenanlage rot zeigt und die Schrankenbäume die untere Endlage erreicht haben. Der Bahnübergang beendet den Sicherungsvorgang, sobald seine Ausschaltsensorik (sen) das Befahren und das Räumen des Bahnübergangsbereichs von dem Eisenbahnfahrzeug detektiert hat. Beim Beenden des Sicherungsvorgangs werden die Schrankenanlage und die Lichtzeichenanlage ausgeschaltet. Der Sicherungsvorgang ist beendet, wenn die Schrankenbäume die obere End-

lage erreicht haben. Es wird verlangt, dass zwischen dem Beenden eines Sicherungsvorgangs und dem Starten des darauf folgenden eine Mindestöffnungszeit ($t_{mö}$) eingehalten wird.

Mit der Abgabe eines Stellbefehls wird die Zeit mitgeteilt, zu der das Eisenbahnfahrzeug beim Bahnübergang eintrifft. Zu dieser Zeit muss sich der Bahnübergang in dem gesicherten Zustand befinden. Der Sicherungsvorgang beginnt mit dem Einschalten der Lichtzeichenanlage. Dabei zeigt die Lichtzeichenanlage zuerst das Gelblicht an, um dem Straßenverkehrsteilnehmer eine bevorstehende Zugfahrt anzukündigen. Nach dem Ablauf der vorgesehenen Gelbzeit (t_G) wird die Lichtzeichenanlage auf Rot umgeschaltet, der Bahnübergang ist nun für die Straßenverkehrsteilnehmer gesperrt. Bevor die Schrankenanlage eingeschaltet wird und die Schrankenbäume zu sinken beginnen, muss die Vorleuchtzeit (t_l) abgelaufen sein. Diese Zeit gibt dem Straßenverkehrsteilnehmer, der sich eventuell noch in dem Bahnübergangsbereich befindet, die Möglichkeit, den Gefahrraum vollständig zu räumen. Nach dem Ablauf der Vorleuchtzeit wird die Schrankenanlage eingeschaltet. Die Schrankenbäume müssen binnen der Schrankenschließzeit (t_s) die untere Endlage erreichen. Ist das nicht der Fall, so lässt sich auf das Vorliegen eines Defektes bei der Schrankenanlage schließen.

Nach dem Erhalt eines Stellbefehls berechnet der Bahnübergang den Einschaltzeitpunk (t_e), zu dem die Lichtzeichenanlage eingeschaltet wird und damit der Sicherungsvorgang beginnt. Der Einschaltzeitpunkt lässt sich mit Hilfe der von dem Eisenbahnfahrzeug mitgeteilten Zeit und unter Berücksichtigung von t_G, t_l und t_s berechnen.

Damit im Fall eines fahrzeugseitigen Fehlers der Bahnübergang nicht weiter für die Straßenverlehrsteilnehmer gesperrt bleibt, ist eine maximale Sperrzeit vorgesehen, die Grundstellungszeit (t_{gs}) genannt wird. Wenn nach dem Start des Sicherungsvorgangs die Grundstellungszeit überschritten ist und das Eisenbahnfahrzeug den Bahnübergang noch nicht erreicht hat, kann ein Fehler beim Eisenbahnfahrzeug die Ursache sein. In diesem Fall wird die Überschreitung der Grundstellungszeit der FFB-Zentrale mitgeteilt. In [ADt97] ist nicht eindeutig beschrieben, wie dieser Fall gehandhabt wird, und ob die FFB-Zentrale dann den Befehl zum Aufheben des Sicherungsvorgangs erteilt. Für die Modellierung wird angenommen, dass der Bahnübergang nach Ablauf der Grundstellungszeit den Sicherungsvorgang beendet.

Das Räumen des Gefahrraums wird von der Ausschaltsensorik des Bahnübergangs detektiert. Danach wird der Sicherungsvorgang beendet. Dabei werden im Modell zuerst die Schrankenanlage und dann die Lichtzeichenanlage ausgeschaltet. Die Schrankenbäume haben binnen der Schrankenöffnungszeit ($t_ö$) die obere Endlage zu erreichen, sonst lässt sich auf einen Defekt bei der Schrankenanlage schließen.

In [ADt97] wird das Verhalten des Bahnübergangs bei den Defekten der Lichtzeichenanlage und der Schrankenanlage nicht eindeutig beschrieben. Diese Fälle wurden in dem Modell, wie folgt, abgebildet:

1. Wenn bei der Lichtzeichenanlage nur das Gelblicht ausgefallen ist, wird direkt auf rot geschaltet und die FFB-Zentrale zwecks Einleitung einer Reparaturmaßname benachrichtigt und der Sicherungsvorgang wird weitergeführt, denn das Rotlicht ist nicht ausgefallen und demzufolge kann der gesicherte Zustand erreicht werden.
2. Bei dem Ausfall des Rotlichts kann der gesicherte Zustand nicht erreicht werden. In diesem Fall wird die FFB-Zentrale benachrichtigt, der Sicherungsvorgang wird abgebrochen und der Bahnübergang geht in den Zustand *Defekt* über. Der Bahnübergang reagiert dann auf jede Statusabfrage bzw. auf jeden Stellbefehl mit der Meldung über den defekten Zustand.
3. Wenn nach dem Starten des Sicherungsvorgangs die Schrankenbäume nach dem Ablauf der Schrankenschließzeit die untere Endlage nicht erreichen, wird der Sicherungsvorgang abgebrochen. Es wird zusammen mit der Benachrichtigung der FFB-Zentrale die bereits eingeschaltete Lichtzeichenanlage ausgeschaltet. Der Bahnübergang geht zwar nicht in den Zustand *Defekt* über, er erreicht aber auch nicht den Zustand *Gesichert*, so dass er auf die fahrzeugseitige Statusabfrage nicht mit der Meldung „gesichert“ reagiert. Das führt dazu, dass das Eisenbahnfahrzeug vor dem Bahnübergang anhält.
4. Wenn beim Beenden des Sicherungsvorgangs die Schrankenbäume die untere Endlage nicht verlassen oder nach dem Ablauf der Schrankenöffnungszeit die obere Endlage nicht erreichen, wird die FFB-Zentrale benachrichtigt und der Bahnübergang geht in den Zustand *Defekt* über. Der Bahnübergang reagiert dann auf jede Statusabfrage bzw. auf jeden Stellbefehl mit der Meldung über den defekten Zustand.

In Abb. 3.6 lässt sich das erläuterte Verhalten von Objekten der Klasse *Bahnübergang* als eine Erweiterung des Verhaltens von Objekten der Klasse *Fahrwegelement* definieren. Die Objekte der Klasse *Bahnübergang* befinden sich zu jeder Zeit in einem der Zustände, die für die Klasse *Fahrwegelement* definiert wurden. Auch die Transitionen zwischen den geerbten Zuständen entsprechen den Transitionen in der Klasse *Fahrwegelement*. So sind z. B. die Transitionen von den Teilzuständen in *Auftrag_Bearbeitend* zum Zustand *Defekt* Erweiterungen der Transition von *Auftrag_Bearbeitend* nach *Defekt* in dem Zustandsdiagramm der Klasse *Fahrwegelement*. In beiden Zustandsdiagrammen werden diese Transitionen durch Feststellung eines Defektes gesteuert. Dabei wird in beiden Zustandsdiagrammen die Aktion *defekt_melden*() ausgeführt. In der Klasse *Bahnübergang* werden bei diesen Transitionen zusätzlich zu *defekt_melden*() weitere jedoch bahnübergangsspezifische Aktionen zur Steuerung der Teilkomponenten ausgeführt, die die Erweiterung des geerbten Verhaltens zeigen.

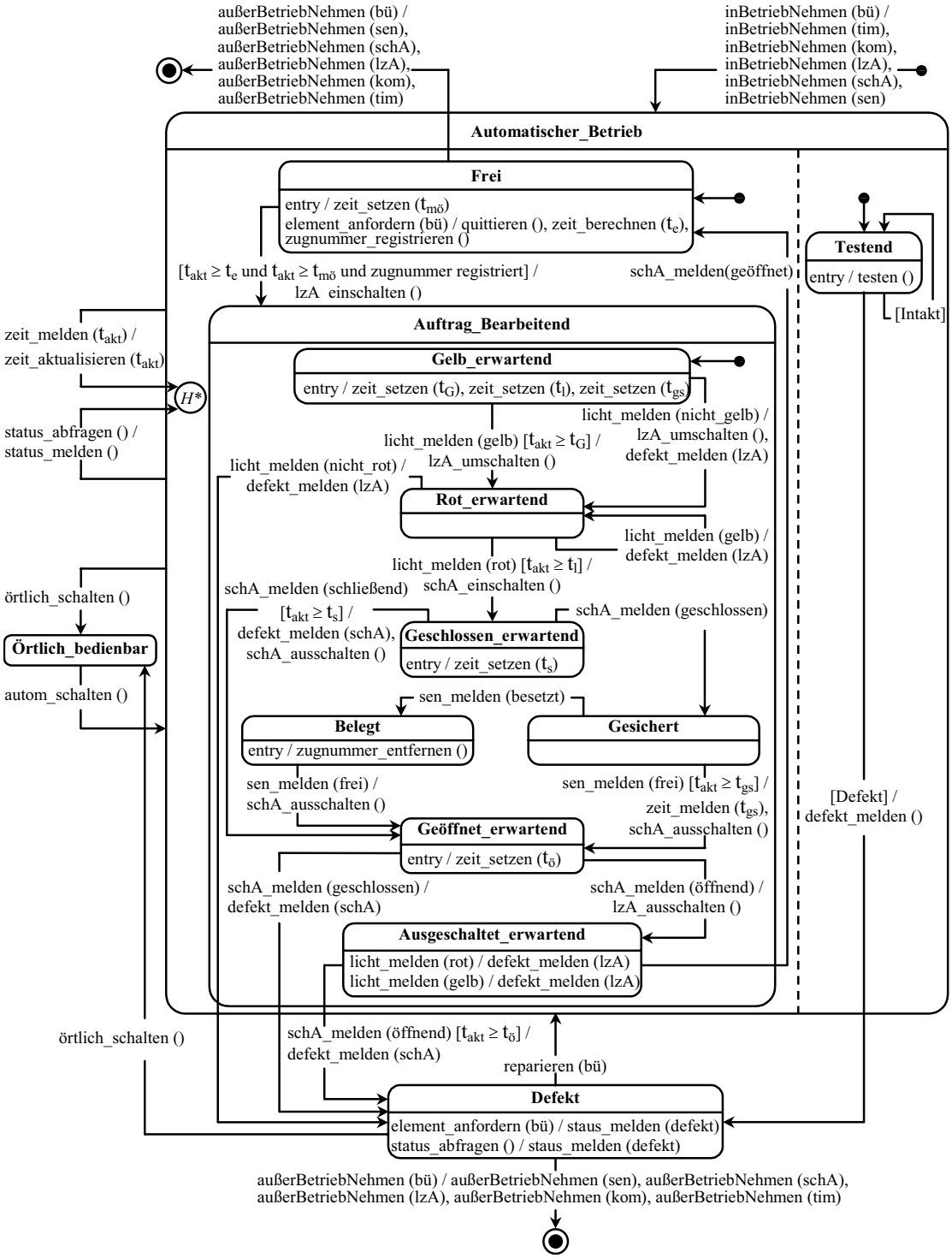


Abb. 3.6 Zustandsdiagramm der Klasse Bahnübergang

Weitere Erweiterungen des Verhaltens sind innerhalb der geerbten Zuständen definiert. In dem Zustand *Frei* werden von einem Objekt der Klasse *Bahnübergang* zusätzlich zur Aktion *quittieren*(), die von der Klasse *Fahrwegelement* geerbt wurde, noch die Aktionen *zeit_setzen*($t_{mö}$) zur Überwachung der Dauer der Mindestöffnungszeit, *zeit_berechen*(t_e) für den Einschaltzeitpunkt und *zugnummer_registrieren* () für die Re-

gistrierung der Identität des Eisenbahnfahrzeugs, das den Stellbefehl abgegeben hat, ausgeführt.

In jedem Sicherungsvorgang werden durch die Aktion *zeit_setzen*() aus der maximalen Zeitdauer für die Mindestöffnungszeit, Gelbzeit, Vorleuchtzeit, Grundstellungszeit und die Schließ- bzw. Öffnungszeit der Schranken aktuelle Werte für die Zeitpunkte $t_{mö}$, t_G , t_l , t_{gs} und t_s bzw. $t_ö$ berechnet, zu denen die genannten Zeiten abgelaufen sind. Bei jeder Ausführung von *zeit_setzen*() wird zur aktuellen Zeit der vorgegebene Wert der entsprechenden Zeitdauer addiert.

Die von der Klasse *Fahrwegelement* übernommene Bedingung [*Startbedingungen erfüllt*] wird von der Klasse *Bahnübergang* in [$t_{akt} \geq t_e$ und $t_{akt} \geq t_{mö}$ und *zugnummer registriert*] überschrieben. In dieser Bedingung stehen das Überschreiten des Einschaltzeitpunkts, der Ablauf der Mindestöffnungszeit und die Registrierung der Zugnummer für die speziellen Startbedingungen, die für die Bahnübergänge gelten. Ferner entspricht die Aktion *lzA_einschalten*(), die das Einschalten der Lichtzeichenanlage veranlasst, der Aktion *sicherung_starten*() bei der Klasse *Fahrwegelement*.

In dem Zustand *Auftrag_Bearbeitend* entspricht das abgebildete Verhalten der Kommunikation mit den Objekten der Klassen *Lichtzeichenanlage*, *Schrankenanlage* und *Ausschaltsensorik*, auf deren Verhalten und Darstellung ihrer Zustandsdiagramme hier nicht weiter eingegangen wird.

In der objektorientierten Modellierung des Anwendungsbeispiels stehen die Beschreibung der Aktivitäten in den Zuständen *Testend* und *Örtlich_Bedienbar* nicht im Vordergrund. Daher wird in diesen Zuständen das Verhalten der Objekte der Klasse Bahnübergang nicht weiter spezifiziert.

3.3.4.1.3 Internes Verhalten der FFB-Zentrale

Da der Schwerpunkt des Anwendungsbeispiels bei dem eingleisigen Bahnübergang im FFB liegt, wird die FFB-Zentrale nicht mit allen in den Lastenheften an sie gestellten Anforderungen in dem Modell abgebildet. Ein Objekt der Klasse *FFB-Zentrale* (zent) ist in dem Modell lediglich für die Zuweisung und Verwaltung der Fahrwege und Fahrwegabschnitte und für die Überprüfung und Erteilung der Fahrerlaubnisse (ferl) zuständig. Dieses Objekt leitet die Meldungen, die bei einer Fehlererkennung an die FFB-Zentrale gesendet werden, an das Objekt der Klasse *Verwaltung* weiter. Die Verwaltung ist dann für die Einleitung der erforderlichen Reparaturmaßnamen zuständig.

In Abb. 3.7 wird das Verhalten für die Klasse *FFB-Zentrale* gezeigt. Dabei kommunizieren die Objekte dieser Klasse mit den Objekten der Klasse *Fahrzeug* und mit ihren eigenen Teilkomponenten Timer (tim), Kommunikationsmodul (kom) und Streckenatlas (stratl).

Mit der Ausführung der Aktion *ferl_anfordern*() sendet ein Objekt der Klasse *Fahrzeug* an das Objekt der Klasse *FFB-Zentrale* die Anforderung nach einer Fahrerlaubnis für

einen bestimmten Fahrweg. Daraufhin wird von dem Objekt der Klasse *FFB-Zentrale* die Aktion *fahrweg_ermitteln*() ausgeführt. Diese Aktion ist eine Suche nach dem gewünschten Fahrweg in dem Teilobjekt Streckenatlas des Objekts der Klasse *FFB-Zentrale*. Das Objekt der Klasse *Streckenatlas* zeigt den Fahrweg an, indem es die Aktion *fahrweg_melden*() ausführt. Durch die Aktion *abschnitt_definieren*() zerlegt dann die Zentrale den angezeigten Fahrweg in Fahrwegabschnitte. Die Zentrale überprüft dann, ob alle definierten Fahrwegabschnitte frei sind, indem sie die Aktion *fahrweg_prüfen*() ausführt. Ist einer der definierten Fahrwegabschnitte vergeben, so wird die Überprüfung wiederholt. Sind alle definierten Fahrwegabschnitte frei, so ist dann der gesamte Fahrweg frei. In diesem Fall wird die Aktion *ferl_senden*() ausgeführt, durch die eine Fahrerlaubnis erteilt wird. Zusammen mit der Ausführung dieser Aktion werden die Aktionen *abschnitt_vergeben*() und *fahrweg_vergeben*() ausgeführt, die den ersten Fahrwegabschnitt und den gesamten Fahrweg als vergeben kennzeichnen.

Die Fahrwegabschnitte, die durch die Fahrerlaubnis an das Objekt der Klasse *Fahrzeug* vergeben worden sind, können von dem Fahrzeug belegt und nach Befahren wieder frei gemeldet werden. Das Objekt der Klasse *Fahrzeug* führt permanent die Aktion *ort_melden*() aus. Damit ist die FFB-Zentrale in der Lage, zu erkennen, welche Fahrwegabschnitte frei, welche belegt und welche noch vergeben und noch nicht freigefahren sind. Mit Hilfe dieser Informationen kann die FFB-Zentrale dann die freigefahrenen Fahrwegabschnitte anderen Fahrzeugen zur Verfügung stellen.

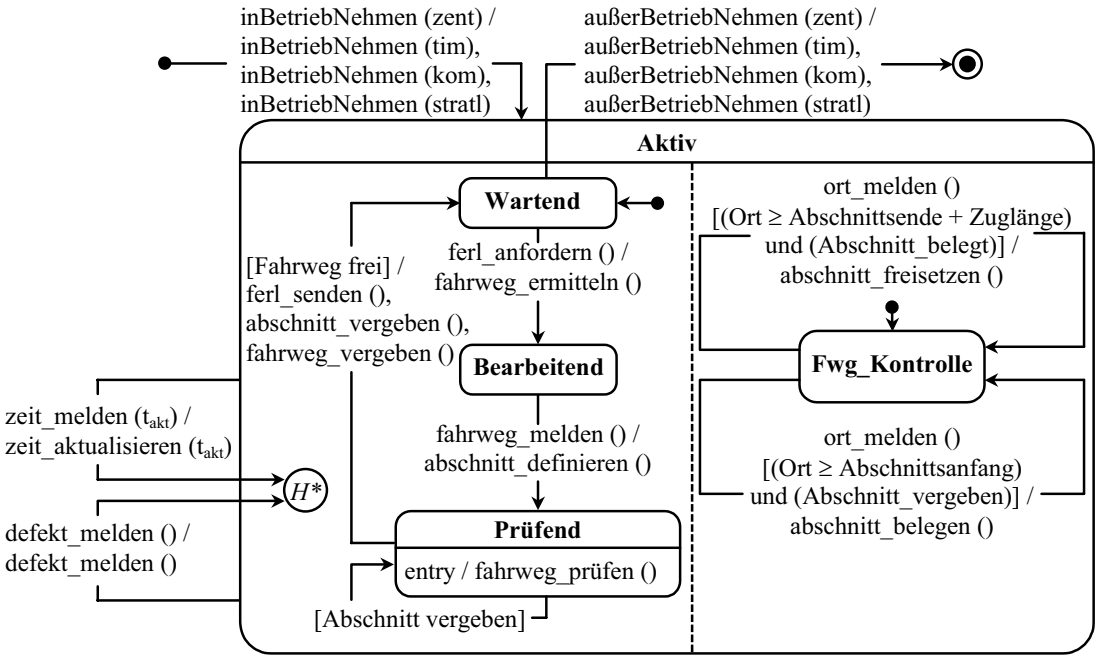


Abb. 3.7 Zustandsdiagramm der Klasse FFB-Zentrale

3.3.4.1.4 Internes Verhalten des Fahrzeugs

Im objektorientierten Modell des Anwendendungsbeispiels kommunizieren die Objekte der Klasse *Fahrzeug* außer mit ihren eigenen Teilkomponenten Kommunikationsmodul, Timer, Streckenatlas, Odometer und Bremse noch mit den Objekten der Klassen *Bahnübergang* und *FFB-Zentrale*. Bei der Modellierung des Verhaltens für die Klasse *Fahrzeug* stehen die Anforderungen im Vordergrund, die für die Steuerung eines eingleisigen Bahnübergangs, für die permanente Ortung und Ortsmeldung bzw. für die Geschwindigkeitsüberwachung relevant sind.

Die Kommunikation mit der FFB-Zentrale besteht in erster Linie darin, eine Fahrerlaubnis (ferl) zu beantragen und den Abfahrtzeitpunkt (t_a) zu bestimmen. Nach der Abfahrt teilt das Fahrzeug permanent seinen aktuellen Ort der FFB-Zentrale mit. Mit Hilfe des aktuellen Orts des Fahrzeugs und seiner Länge ist die FFB-Zentrale in der Lage, die freigefahrenen Fahrwegabschnitte zu identifizieren, und diese einem weiteren Fahrzeug zuzuweisen.

Die Kommunikation mit dem Bahnübergang besteht darin, ihm einen Stellbefehl zur Sicherung des Bahnübergangs zu erteilen, und eine Statusabfrage zu stellen. Kann der Bahnübergang sich nicht selbstständig sichern, muss die Sicherung manuell erfolgen, um so die Weiterfahrt zu ermöglichen. Für diese Kommunikation mit dem Bahnübergang berechnet das Fahrzeug entsprechend seiner aktuellen Geschwindigkeit und seinem aktuellen Ort (Ort) einen Zeitpunkt zur Aufnahme des Kontakts mit dem Bahnübergang (Bü). Zu diesem Zeitpunkt (kzp) wird mit dem Bahnübergang der Kontakt aufgenommen und der Stellbefehl abgegeben. Nach der Abgabe des Stellbefehls wird die maximale Zeit bis zum Erhalt einer Quittung (t_q) berechnet. Quittiert der Bahnübergang innerhalb von t_q den Erhalt des Stellbefehls, so wird der Funkkontakt mit dem Bahnübergang beendet, und ein weiterer Kontaktzeitpunkt bestimmt. Der zweite Kontaktzeitpunkt ist für die Statusabfrage vorgesehen. Befindet sich der Bahnübergang bei der Statusabfrage in dem gesicherten Zustand, so kann das Fahrzeug unbehindert den Bahnübergang passieren und den nächsten Gefahrpunkt suchen, der z. B. wieder ein Bahnübergang oder der Endpunkt des zugewiesenen Fahrwegs sein kann. Erhält das Fahrzeug innerhalb von t_q keine Quittung bzw. trifft das Fahrzeug innerhalb der Grundstellungszeit (t_{gs}) nicht am Bahnübergang ein, oder reagiert der Bahnübergang auf die Statusabfrage nicht mit der Meldung über das Erreichen des gesicherten Zustands, so hat das Eisenbahnfahrzeug vor dem Bahnübergang anzuhalten ([V = 0]). In diesem Fall erfolgt die Sicherung des Bahnübergangs vor Ort durch das Personal. Dabei wird der Bahnübergang durch die Ausführung der Aktion *örtlich_schalten*(), die dem Bedienen einer entsprechenden Taste entspricht, in den Zustand *Örtlich_Bedienbar* versetzt. Nachdem das Eisenbahnfahrzeug den Bahnübergangsbereich vollständig geräumt hat, wird der Bahnübergang wiederum durch das Personal in den Zustand *Automatischer_Betrieb* zurückgesetzt.

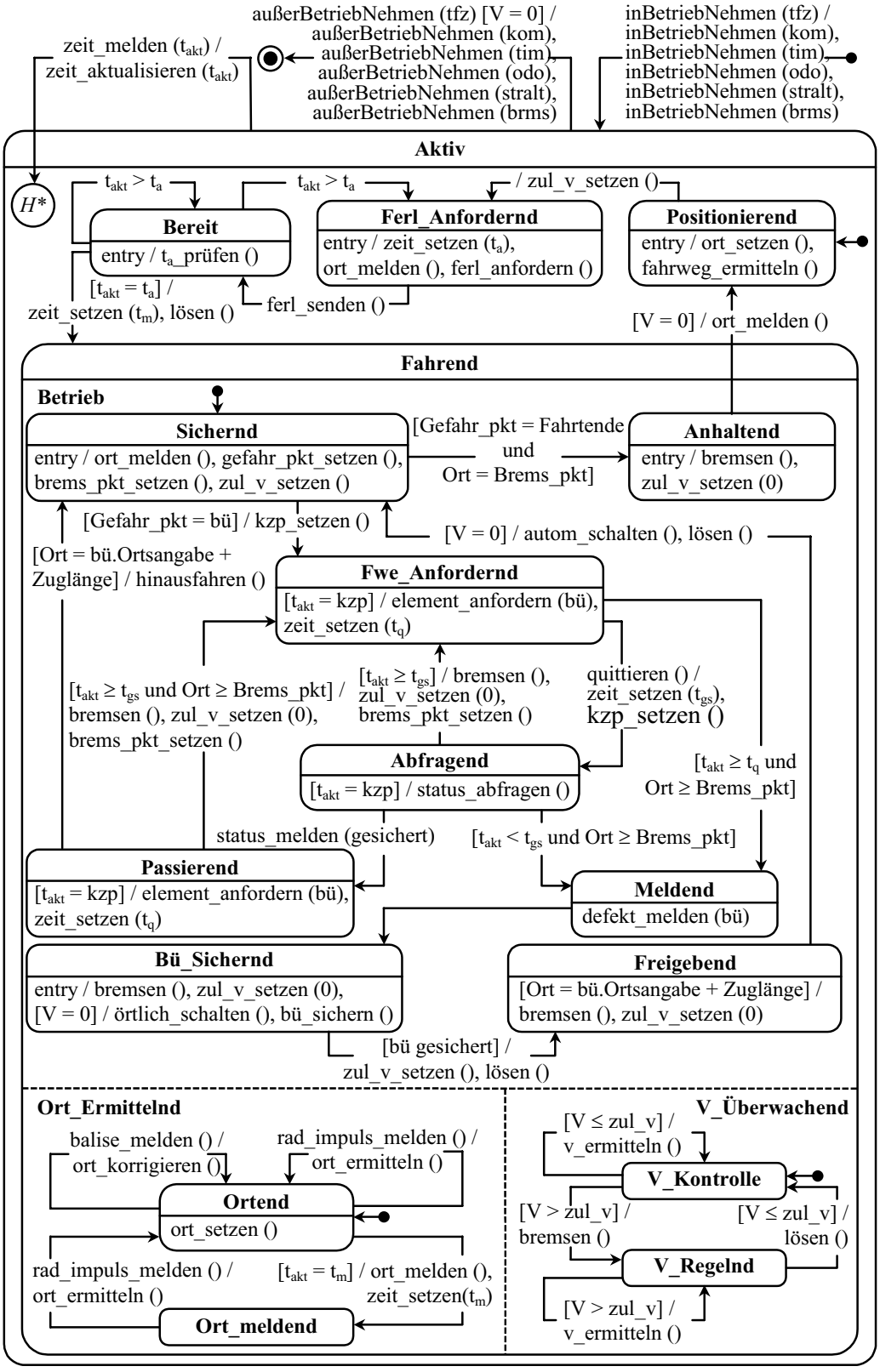


Abb. 3.8 Zustandsdiagramm der Klasse Fahrzeug

Die permanente Ortung erfolgt durch eine Kommunikation mit der Teilkomponente Odometer (odo) und durch die Ausführung der Aktion *ort_setzen*(). Als Reaktion auf das Ereignis *rad_impuls_melden*(), das von der Teilkomponente Odometer ausgelöst wird, führt das Objekt der Klasse *Fahrzeug* die Aktion *ort_ermitteln*() aus und dadurch wird die aktuelle Ortsangabe ermittelt. Die permanente Ortsmeldung an die FFB-Zentarle findet in einem vorgegebenen zeitlichen Abstand von t_m durch die Ausführung der Aktion *ort_melden*() statt. Sobald das Objekt der Klasse *Fahrzeug* über eine Balise gefahren ist und durch das Ereignis *balise_melden*() über die absolute Ortsangabe verfügt, korrigiert es die berechnete Ortung durch die Ausführung der Aktion *ort_korrigieren*().

Die Geschwindigkeitsüberwachung findet durch permanente Ermittlung der aktuellen Geschwindigkeit (V), die durch die Ausführung der Aktion *v_ermitteln*() passiert, und durch das permanente Vergleichen der aktuellen Geschwindigkeit mit der zulässigen Geschwindigkeit (zul_v) statt. Die Aktionen *bremsen*() und *lösen*() steuern die Teilkomponente Bremse und beeinflussen auch die aktuelle Geschwindigkeit.

Die permanente Ortung sowie die Geschwindigkeitsüberwachung erfolgen unabhängig von weiteren Aktivitäten des Fahrzeugs. Daher werden diese Aktivitäten parallel zu anderen Aktivitäten des Fahrzeugs modelliert. Diese Aktivitäten beginnen, sobald sich das Fahrzeug durch die Aktion *lösen*(), die die Bremsen löst, in Bewegung setzen kann und enden unter der Bedingung [V = 0], die den Stillstand des Fahrzeugs bedeutet. Das dynamische Verhalten der Objekte der Klasse Fahrzeug ist in Abb. 3.8 gezeigt worden.

Nach der Beschreibung des internen Verhaltens der Hauptkomponenten FFB-Zentrale, Fahrzeug und Bahnübergang, werden die Anforderungen bezüglich der Interaktionen unter diesen Komponenten erläutert.

3.3.4.2 Interaktionen von Systemkomponenten im FFB

Im Anwendungsbeispiel soll die Aufgabe der Fahrwegsicherung durch eine Kooperation zwischen den Fahrzeugen und den zustandsvariablen Fahrwegelementen realisiert werden. Das Fahrzeug sendet rechtzeitig einen Stellbefehl an das Fahrwegelement, der von dem Fahrwegelement quittiert werden soll. Nach dem Abschluss des Sicherungsvorgangs findet eine fahrzeugseitige Statusabfrage statt, auf die seitens des Fahrwegelements mit einer Meldung über den aktuellen Zustand reagiert wird. Entsprechend der Statusmeldung wird entschieden, ob das Fahrzeug das Fahrwegelement befahren darf oder vor ihm zum Halten kommen muss.

Die in [ADt97] dargestellten Abläufe lassen sich in Form von Interaktionen unter den Betriebskomponenten darstellen. In dem objektorientierten Modell werden für die Beschreibung der Interaktionen und Systemprozesse die Sequenzdiagramme der UML verwendet. In einem Sequenzdiagramm (Abb. 3.9, 3.10) werden die an einem Prozess beteiligten Objekte in Vierecken dargestellt. Für jedes Objekt wird eine senkrechte Linie gezeichnet, die den temporären Ablauf von Informationen darstellt, die von dem

Objekt gesendet oder empfangen werden. Die Informationen selbst werden in Form von Pfeilen dargestellt, die von einem Objekt zu einem anderen Objekt gezeichnet werden.

Eine Zugfahrt erfolgt, indem das Fahrzeug bei der FFB-Zentrale für die zu befahrende Strecke eine Fahrerlaubnis beantragt. Ist die Strecke frei, so wird die Fahrerlaubnis erteilt. Befindet sich auf der Strecke ein Bahnübergang, so berechnet das Fahrzeug den Zeitpunkt, an dem zwecks Erteilung eines Stellbefehls Kontakt aufgebaut werden soll. Zu dem Kontaktzeitpunkt ($[t_{akt} \geq kzp]$) sendet das Fahrzeug einen Stellbefehl an den Bahnübergang. Der Bahnübergang quittiert den Erhalt des Stellbefehls und dieser hat dann die Aufgabe, sich zu sichern. Der Bahnübergang darf nur befahren werden, wenn das Fahrzeug die Meldung über den gesicherten Zustand des Bahnübergangs erhalten hat. Nachdem das Fahrzeug den Bahnübergang passiert hat, wird der Sicherungsvorgang beendet.

In Abb. 3.9 wird dieses Szenario unter der Beteiligung der Objekte der Klassen *FFB-Zentrale*, *Fahrzeug* und *Bahnübergang* dargestellt. Dieses Szenario beschreibt den fehlerfreien Fall, in dem weder beim Fahrzeug ein Defekt oder eine Verspätung vorliegt noch der Bahnübergang oder einer seiner Teile defekt sind.

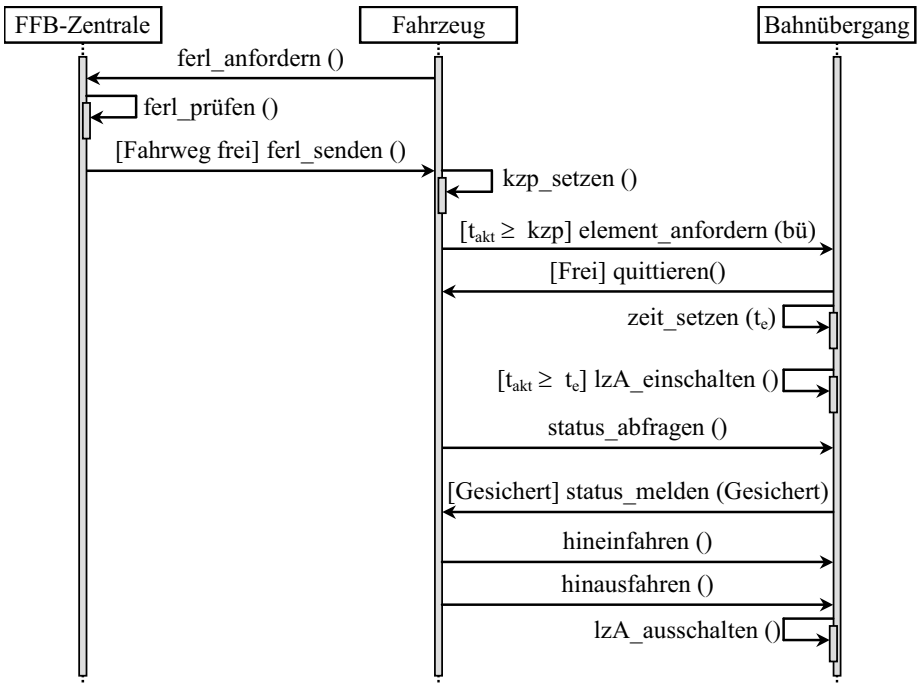


Abb. 3.9 Interaktionen unter den Systemkomponenten in dem fehlerfreien Fall

Liegt ein Fehler vor, so dass das Fahrzeug die Meldung über den gesicherten Zustand des Bahnübergangs nicht erhält, so hat das Fahrzeug am vorher berechneten Bremseinsatzpunkt (Brems_pkt) die Bremsen zu betätigen, um vor dem Bahnübergang zum Halten zu kommen. In diesem Fall muss der Bahnübergang, wie bereits erwähnt, vor Ort vom Personal gesichert werden. Nachdem das Fahrzeug den Bahnübergang nach

einer manuellen Sicherung passiert hat, wird der Bahnübergang in dem automatisch zu betreibenden Zustand zurückgeschaltet.

Dieser Fall kann vorkommen, wenn aufgrund eines Fehlers beim Bahnübergang oder bei einem seiner Teile der gesicherte Zustand nicht erreicht werden kann, oder wenn aufgrund des ausgefallenen Kommunikationsmoduls der Kontakt zwischen dem Fahrzeug und dem Bahnübergang nicht zustande kommt und der Bahnübergang den Stellbefehl nicht erhält bzw. er den Erhalt des Stellbefehls nicht quittiert. Für die Beschreibung jedes dieser Fälle wird ein separates Sequenzdiagramm benötigt. In Abb. 3.10 wird beispielhaft das Szenario gezeigt, in dem der Bahnübergang defekt ist jedoch die Kommunikation zwischen dem Fahrzeug und dem Bahnübergang stattfindet.

Die in den letzten Abschnitten erstellte Systemanforderungsdefinition umfasst nun die Beschreibung der Systemstruktur und die Beschreibung des Systemverhaltens. Derartige Systemdefinitionen sind im Vergleich zu einer in einer natürlichen Sprache verfassten Systemdefinition präziser und erfüllen die folgenden Anforderungen:

1. Möglichkeit zur Abbildung unterschiedlicher Sichtweisen auf das System durch die Abbildung der Systemstruktur, des Verhaltens einzelner Systemkomponenten und der Interaktionen innerhalb des Systems,
2. Modularität durch Kapselung einzelner Systemkomponenten und Kompositionalität durch die Spezifikation der Verhältnisse zwischen den Systemkomponenten,
3. Rückführbarkeit und Identifizierung einzelner Module durch die Darstellung der Systemkomponenten in Form von Objekten bzw. Klassen,
4. Wiederverwendbarkeit und durchgängige Anwendung im gesamten Entwicklungs- und Zulassungsprozess, denn bei den weiteren Entwicklungsphasen kann die Beschreibung der Systemkomponenten und ihrer Verhältnisse ohne eine erneute Beschreibung lediglich weiterverfeinert werden,
5. Validierbarkeit und Ausführbarkeit im Sinne einer Simulation des künftigen Systems. Diese Anforderung wird durch verschiedene UML-Werkzeuge unterstützt, indem eine Simulation auf Basis der in der Definitionsphase erstellten Diagramme durchgeführt werden kann.
6. Vereinfachte Handhabung bei Änderungen. Diese Anforderung wird durch den modularen Aufbau der Systemdefinition erfüllt. Jede Änderung muss lediglich an den betreffenden Objekten durchgeführt werden.
7. Einfache Erlernbarkeit, Handhabbarkeit und Interpretierbarkeit für die Eisenbahningenieure. Durch die grafische Gestallt und Beschränkung der Anzahl grafischer Symbole sind die angewendeten Beschreibungsmittel leicht erlernbar und handhabbar.

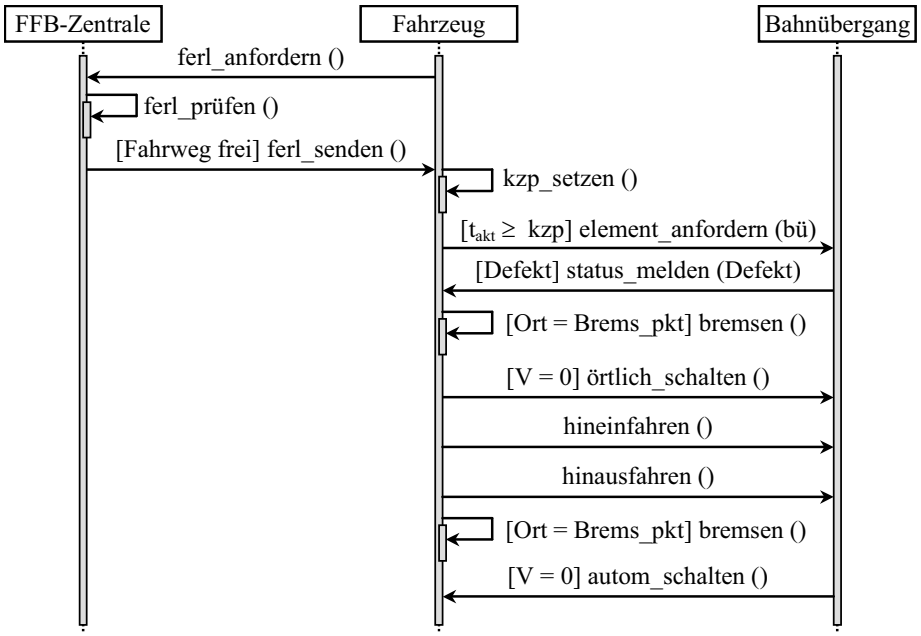


Abb. 3.10 Systemablauf mit dem defekten Bahnübergang

Die Anforderung nach Eindeutigkeit und Interpretierbarkeit setzt eine Präzisierung der angewendeten Beschreibungsmittel voraus. Dabei sollen die Beschreibungsmittel und ihre Anwendung so definiert werden, dass sie eine eindeutige Bedeutung haben. Diese eindeutige Bedeutung soll auf den Eigenschaften der zu spezifizierenden Objekte und Sachverhalte basieren und die Ingenieure bei der Erstellung und Interpretation eines eindeutigen Modells unterstützen. Mit diesem Ziel werden zunächst in Kapitel fünf die Diagrammarten, die zur Beschreibung der Systemdynamik verwendet werden, formal definiert.

Um die Anforderung nach vereinfachten Änderungen zu erfüllen, müssen die Relationen, die zur Beschreibung der Beziehungen unter den Systemkomponenten verwendet werden, präzise untersucht und eindeutig definiert werden. Eine formale Definition dieser Relationen unterstützt die Durchführung der Änderungen und der eventuellen Korrekturen im Modell. Anhand dieser Definitionen kann festgelegt werden, ob eine Änderung in der Struktur bzw. im Verhalten eines Objekts entsprechende Änderungen bei anderen Objekten im System zur Folge hat. In Kapitel sechs werden der Aufbau eines Objektsystems, die Beschreibung der Systemstruktur und die Relationen unter den Systemkomponenten formal definiert.

Damit die Anforderung nach Einbindung formaler Verifikation erfüllt wird und überprüft werden kann, ob das Modell die funktionalen Sicherheitsanforderungen erfüllt, die an das System gestellt werden, müssen entsprechende Modellteile in eine formal verifizierbare Form transformiert werden. Kapitel sieben befasst sich mit diesem Aspekt. Dabei wird mit Hilfe der Definitionen in Kapitel fünf eine Transformation des UML-

Modells in eine formal verifizierbare Form präsentiert. Weiter wird in diesem Kapitel eine formale Darstellung der zu überprüfenden Sicherheitsanforderungen dargestellt.

4 Formale Systemspezifikation

In Kapitel vier werden Systemspezifikation und Anforderungen an ein System behandelt. In diesem Zusammenhang kann der Begriff formal verifizierte Systemanforderungsspezifikation definiert werden.

Die Grundlage einer Systemanforderungsspezifikation ist in der vorliegenden Arbeit ein Objektsystem, dessen Grundbegriffe wie Objekte und Klassen hier definiert werden. Die Elemente zur Beschreibung von Objekten und Klassen, wie z. B. Attribute, Aktionen, Ereignisse etc. werden ebenso in diesem Kapitel behandelt.

Das Ziel dieses Kapitels ist, für die Grundbegriffe eines Objektssystems, die in weiteren Kapiteln der Arbeit angewendet werden, Bezeichnungen und geeignete Notationen einzuführen.

4.1 Systemanforderungsspezifikation

Eine Systemanforderungsspezifikation ist eine Sammlung von Anforderungen an ein System. Diese Anforderungen beziehen sich auf den Aufbau, die Beschaffenheit, die Struktur und das Verhalten des Systems. Weiter beschreiben Anforderungen die Rahmenbedingungen für die Entwicklung bzw. für den Einsatz des Systems. Damit das System im späteren Betrieb auch unter den äußeren Einflüssen die erwartete Funktionalität

erbringt, werden auch Anforderungen an die Systemumgebung in der Systemanforderungsspezifikation aufgenommen.

Zu einer Systemanforderungsspezifikation gehören im Allgemeinen auch Anforderungen bezüglich der Entwicklungskosten, Wartung, Instandhaltung sowie der Hinweis auf Gesetze und Vorschriften, die bei der Entwicklung und beim Betrieb eingehalten werden müssen.

Unter den genannten Anforderungen, die jeweils ein Aspekt des Systems beschreiben, sind die Anforderungen zur Spezifikation der Systemstruktur und des dynamischen Systemverhaltens die Charakterisierenden. Die Bedeutung dieser Anforderungen wird in [Par98] wie folgt beschrieben:

Ein System ist charakterisiert durch seine Struktur und sein Verhalten. Dabei fasst die Systemstruktur die Gesamtheit der Komponenten und ihrer Beziehungen zusammen und das Systemverhalten das Zusammenwirken der Komponenten zur Erreichung des Systemziels oder -zwecks.

In der vorliegenden Arbeit bezeichnet die Systemanforderungsspezifikation die Spezifikation von Anforderungen bezüglich der Struktur und des dynamischen Verhaltens eines Systems. Dabei ist die Systemstruktur die Systemkomponenten und ihre Beziehung zueinander und das Systemverhalten das dynamische Verhalten einzelner Systemkomponenten und die Interaktionen, die zwischen den Systemkomponenten und mit der Systemumgebung stattfinden.

Beim objektorientierten Ansatz zur Beschreibung von Systemen wird ein System als eine Menge interagierender Objekte betrachtet. Hier werden für die Beschreibung der Systeme zwei Mengen eingeführt, die die Systemkomponenten und die Anforderungen an das System beinhalten.

4.2 Objektsystem

Ein Objektsystem ist ein geordnetes Paar $S = (M, R)$, wobei M eine nichtleere und eventuell endliche Menge von miteinander kommunizierenden Objekten und R eine nichtleere Menge von Anforderungen ist.

Für eine Menge von Bedingungen B mit $B \subseteq R$ drückt die Schreibweise $S \vDash B$ aus, dass das Objektsystem S die Elemente von B erfüllt. In diesem Fall ist S bezüglich B verifiziert.

Die Aussage $S \vDash B$ ist dann korrekt, wenn in einem Verfahren für jedes Element $b \in B$ gezeigt wird, dass $S \vDash b$ gilt. Dieses Verfahren heißt Verifikation und wird in Kapitel sieben näher ausgeführt.

4.2.1 Beispiel

In dem objektorientierten Modell des FFB besteht das Objektsystem aus FFB-Zentralen, Fahrzeugen, Fahrwegelementen und Verwaltung, die per Funk miteinander kommunizieren und das gesamte Spektrum der Leistungsanforderungen im Bahnbetrieb abdecken. Dabei besteht die Menge M aus allen FFB-Zentralen, Fahrzeugen, Fahrwegelementen, Verwaltung und ihren Teilkomponenten. R ist die Menge aller Anforderungen an das System. Die Elemente von R sind Anforderungen bezüglich der Funktionalität und des Aufbaus und Anforderungen bezüglich der Einhaltung der gesetzlichen, sicherheitsrelevanten und betrieblichen Rahmenbedingungen.

4.3 Abstraktionen in der objektorientierten Analyse

Bei der objektorientierten Analyse für die Modellierung eines Systems trägt die Abstraktion einen wesentlichen Beitrag zur Wiederverwendung der spezifizierten Komponenten und dadurch zur Vermeidung der Redundanzen in der Systemanforderungsspezifikation bei.

Mit der Abstraktion wird die Spezifikation der Gemeinsamkeiten unter den Systemkomponenten in Bezug auf ihre Eigenschaften und ihre Funktionalitäten bezeichnet. Dadurch werden die Typen bzw. die Klassen festgelegt, die die allgemeinen Eigenschaften von einzelnen Objekten beschreiben. In der Literatur zur objektorientierten Analyse werden diese Begriffe nicht einheitlich verwendet. Laut [Par98] dient ein Typ der Spezifikation des externen Verhaltens von Objekten und wird durch eine Klasse realisiert. In [Rum96] werden Typ und Klasse als syntaktische Begriffe bezeichnet. Dabei beschreibt die Klasse die Eigenschaften ihrer Objekte (Agenten) und der Typ spezifiziert die Eigenschaften, die über eine Klassenhierarchie gelten.

In dieser Arbeit wird diese Art von Klassifizierung nicht verwendet. Hier werden die Begriffe Objekt und Klasse benutzt. Dabei wird jede bestimmte Systemkomponente als ein Objekt bezeichnet. Für die Beschreibung aller gleichartigen Systemkomponenten wird die Klasse verwendet.

Jedes Objekt des Objektsystems hat Attribute, reagiert auf die Ereignisse, führt Aktionen aus und hat Beziehungen zu anderen Objekten des Objektsystems. Diese Eigenschaften lassen sich, wie folgt, formal spezifizieren.

4.4 Definition (Objekte und Klassen)

Ein Objekt O in M wird durch das Tupel $O = (\mathcal{A}ttr_O , \mathcal{E}_O , \mathcal{B}_O , \mathcal{A}_O , \mathcal{M}_O , \mathcal{R}_O)$ beschrieben. Dabei sind

- $\mathcal{A}ttr_O$ eine endliche nichtleere Menge von Attributen mit
  $\mathcal{A}ttr_O = \mathcal{A}ttr_O^{st} \cup \mathcal{A}ttr_O^{dy}$ und $\mathcal{A}ttr_O^{st} \cap \mathcal{A}ttr_O^{dy} = \varnothing$,
- $\mathcal{E}_O$ eine endliche nichtleere Menge von Ereignissen,
- $\mathcal{B}_O$ eine endliche nichtleere Menge von Bedingungen,
- $\mathcal{A}_O$ eine endliche nichtleere Menge von Aktionen,
- $\mathcal{M}_O$ eine Zustandsmaschine bzw. ein Zustandsdiagramm und
- $\mathcal{R}_O$ eine endliche Menge von Relationen.

$\mathcal{A}ttr_O$ enthält die Beschreibung aller Attribute des Objekts. Dabei ist die Teilmenge $\mathcal{A}ttr_O^{st}$ die Menge aller statischen Attribute. Dazu gehört in erster Linie der Objektname, durch den sich jedes Objekt von den anderen Objekten des Objektsystems unterscheidet. Die weiteren Elemente von $\mathcal{A}ttr_O^{st}$ sind Parameter zur Bezeichnung der statischen Eigenschaften des Objekts, die im Laufe des Lebenszyklus des Objekts unverändert bleiben. Die Teilmenge $\mathcal{A}ttr_O^{dy}$ enthält hingegen die dynamischen bzw. die variablen Attribute, deren Werte im Laufe des Lebenszyklus des Objekts mittels Aktionen geändert werden und das dynamische Verhalten des Objekts gestallten.

$\mathcal{E}_O$ ist die Menge von Ereignissen, auf die das Objekt *O* in Form von Ausführung von Aktionen oder einem Zustandswechsel reagiert. Die Elemente von $\mathcal{E}_O$ können das dynamische Verhalten von *O* steuern. Zu $\mathcal{E}_O$ gehören Ergebnisse von Aktionen oder einfach Aktionen, die von den mit *O* kommunizierenden Objekten durchgeführt werden.

Da in UML im Kontext der Aktivitäten eines Objekts die Angabe eines Ereignisses optional ist, wird in der vorliegenden Arbeit von einem leeren Ereignis gesprochen, wenn das Objekt für seine bevorstehende Aktivität auf kein Ereignis warten muss. Das leere Ereignis wird hier mit e_ε bezeichnet.

$\mathcal{B}_O$ ist die Menge von Bedingungen, die in Form von logischen Ausdrücken und meistens im Kontext der Attributswerte von *O* oder von Objekten formuliert werden, die mit *O* kommunizieren. In UML ist im Kontext der Aktivitäten eines Objekts die Angabe einer Bedingung optional. Daher wird hier von einer leeren Bedingung gesprochen, wenn eine Aktivität von keiner Bedingung abhängig ist. Die leere Bedingung wird hier mit b_ε bezeichnet und ist immer wahr.

Um eine Aktion im Kontext ihres auslösenden Ereignisses und ihrer Vorbedingung zu kennzeichnen, wird die Schreibweise *e* [*b*] / *a* mit $e \in \mathcal{E}_O$, $b \in \mathcal{B}_O$ und $a \in \mathcal{A}_O$ verwendet. Diese bedeutet, dass *a* ausgeführt wird, wenn *e* eintritt und *b* erfüllt ist.

$\mathcal{A}_O$ ist die Menge der Aktionen, deren Elemente von dem Objekt *O* durchgeführt werden. Dabei nimmt *O* Zustände an, die seinen Lebenszyklus beschreiben. Die Aktionen

eines Objekts können aufgrund des Auftretens von Ereignissen oder in Abhängigkeit von Bedingungen ausgeführt werden.

Ebenso wie bei Ereignissen und Bedingungen ist in UML die Angabe einer Aktion optional. Hier wird von einer leeren Aktion gesprochen, wenn das Objekt, ohne eine Aktion durchzuführen, einen Zustandswechsel unternimmt, oder auf das Eintreten eines Ereignisses wartet. Die leere Aktion wird mit a_ε bezeichnet. Sie verbraucht keine Zeit und hat keine Auswirkung auf die Werte der Attribute.

Für ein Objekt O_i gilt $\mathcal{A}_{O_i} = \{ a_\varepsilon \}$, wenn O_i in seinem Lebenszyklus keine Aktionen ausführt.

$\mathcal{M}_O$ ist eine Zustandsmaschine bzw. ein Zustandsdiagramm zur Darstellung des dynamischen Verhaltens von O. In $\mathcal{M}_O$ wird beschrieben, wie die Aktionen des Objekts O in Abhängigkeit von Ereignissen und Bedingungen ausgeführt werden, bzw. welche Zustände das Objekt O in seinem Lebenszyklus annimmt. Eine genaue Definition einer Zustandsmaschine wird später folgen.

$\mathcal{R}_O$ umfasst Elemente, welche die Relationen zwischen O und anderen Objekten des Objektsystems beschreiben. Die Elemente von $\mathcal{R}_O$ enthalten Angaben bezüglich der Identität der Kommunikationspartner von O sowie Angaben über die Struktur und den Aufbau von O. Aus jedem Element in $\mathcal{R}_O$ ist die Art der Beziehung zwischen O und einem anderen Objekt des Objektsystems zu entnehmen.

Die Elemente in $\mathcal{R}_O$ haben die Form O *rel* O_i oder O_i *rel* O , wobei *rel* die Relation zwischen O und O_i ist. Die Relationen zwischen Objekten sind Assoziation, Aggregation, Komposition und Vererbung. Auf diese Relationen und auf die Elemente von $\mathcal{R}_O$ werden in Kapitel sechs ausführlich eingegangen.

Die Objekte in M werden mit O_1 , O_2 , O_3 , ... bezeichnet. Zwei Objekte gehören zu einer Klasse, wenn alle ihre Eigenschaften identisch sind. Bei den Objekten einer Klasse werden die entsprechenden Attribute durch gleiche Datentypen beschrieben. Ebenso die Aktionen, die Ereignisse und die Bedingungen haben gleiche Beschreibung. Der statische Aufbau aller Objekte einer Klasse ist identisch und ihr dynamisches Verhalten wird durch die gleiche Zustandmaschine spezifiziert.

Die Klassen werden mit K^1, K^2, K^3, ... und ein Objekt O_i aus der Klasse K^p wird mit $K^p_{O_i}$ bezeichnet. Jede Klasse repräsentiert Objekte mit identischem Aufbau und Verhalten. Das heißt, hat ein Objekt einer Klasse eine bestimmte Anzahl von Komponenten, so haben alle Objekte dieser Klasse die gleiche Anzahl von den Komponenten. Und wenn ein Objekt einer Klasse mit einem Objekt aus einer anderen Klasse kommuniziert, so kommunizieren alle Objekte seiner Klasse mit den entsprechenden Objekten aus derselben Klasse. Diese Eigenschaft wird bei der Beschreibung von Relationen unter den Objekten genauer spezifiziert.

In Zusammenhang mit einer Klasse K^p werden die Mengen der Attribute, der Ereignisse, der Bedingungen, der Aktionen und der Relationen sowie die Zustandsmaschine, die das Verhalten der Objekte dieser Klasse beschreibt, mit $\mathcal{A}ttr_{K^p}$, $\mathcal{E}_{K^p}$, $\mathcal{B}_{K^p}$, $\mathcal{A}_{K^p}$, $\mathcal{R}_{K^p}$ und $\mathcal{M}_{K^p}$ bezeichnet.

Im Folgenden wird am Beispiel der Klasse *Lichtzeichenanlage* der Unterschied zwischen einer Klasse und ihrer Objekte bzw. der Unterschied zwischen den genannten Mengen hinsichtlich einer Klasse und ihrer Objekte gezeigt.

4.4.1 Beispiel

Im FFB ist ein bestimmter Bahnübergang $Bü_1$ ein Objekt. Die Klasse *Bahnübergang* beschreibt alle Bahnübergänge vom gleichen Typ. Für die Klasse *Bahnübergang* gilt:

Bahnübergang =

($\mathcal{A}ttr_{Bahnübergang}$, $\mathcal{E}_{Bahnübergang}$, $\mathcal{B}_{Bahnübergang}$, $\mathcal{A}_{Bahnübergang}$, $\mathcal{M}_{Bahnübergang}$, $\mathcal{R}_{Bahnübergang}$) mit

- $\mathcal{A}trr^{st}_{Bahnübrgang}$ = {ID = *string*},
- $\mathcal{A}trr^{dy}_{Bahnübergang}$ = {Frei = *boolean*, $t_{mö}$ = *integer*, t_{akt} = *integer*, …},
- $\mathcal{E}_{Bahnübergang}$ = {*element_anfordern*(*bü*), *licht_melden*(*rot*), ... },
- $\mathcal{B}_{Bahnübergang}$ = {$t_{akt} \geq t_{mö}$, $t_{akt} \geq t_e$, zugnummer registriert, ... },
- $\mathcal{A}_{Bahnübergang}$ = { *quittieren*(), *lzA_einschalten*(), *schA_einschalten*(), ...},
- $\mathcal{M}_{Bahnübergang}$ wurde in Abb. 3.6 gezeigt und
- $\mathcal{R}_{Bahnübergang}$ = {*Bahnübergang comp Timer*, *Bahnübergang asso Fahrzeug*, ... }.

Die Zustandsmaschine $\mathcal{M}$ wird in Kapitel fünf und die Menge der Relationen $\mathcal{R}$ wird in Kapitel sechs erläutert.

4.5 Erzeugen und Löschen von Objekten

Um die Inbetriebnahme bzw. die Außerbetriebnahme von Systemkomponenten zu modellieren, werden Aktionen zum Erzeugen bzw. zum Löschen von Objekten eingeführt. Diese Aktionen können von den bereits vorhandenen Objekten des Objektsystems durchgeführt werden, um Objekte einzuführen bzw. zu entfernen.

Wenn das Objekt $K^p_{O_i}$ die Aktion $a_{init}(K^q_{O_i}) \in \mathcal{A}_{K^p}$ bzw. $a_{fin}(K^q_{O_i}) \in \mathcal{A}_{K^p}$ ausführt, so wird das Objekt $K^q_{O_i}$ erzeugt bzw. gelöscht. In $\mathcal{E}_{K^q}$ der Menge der Ereignisse der Klasse K^q

existieren entsprechend die Elemente $e_{init}(K_{O_i}^q)$ bzw. $e_{fin}(K_{O_i}^q)$ mit $e_{init}(K_{O_i}^q) = a_{init}(K_{O_i}^q)$ bzw. $e_{fin}(K_{O_i}^q) = a_{fin}(K_{O_i}^q)$.

Die Aktionen a_{init} und a_{fin} heißen Initial- und Finalaktion und werden von einem Objekt ausgeführt, um ein anderes Objekt zu initialisieren bzw. in den Endzustand zu versetzen. Die Ereignisse e_{init} bzw. e_{fin} heißen Initial- bzw. Finalereignis und versetzen das entsprechende Objekt in den Initial- bzw. in den Endzustand.

4.5.1 Beispiel

Für ein Objekt *Bü* der Klasse *Bahnübergang* entsprechen a_{init}(*bü*) bzw. e_{init}(*bü*) der Aktion *inBetriebNehmen*(*bü*). Damit wird das Objekt *bü* erzeugt, wenn ein Objekt der Klasse *Verwaltung* die Aktion *inBetriebNehmen*(*bü*) ausführt. Diese Aktion gehört zu $\mathcal{A}_{verwaltung}$ der Menge der Aktionen der Klasse *Verwaltung* und zu $\mathcal{E}_{Bahnübergang}$ derr Menge der Ereignisse der Klasse *Bahnübergang*. Ebenfalls entsprechen a_{fin}(*bü*) und e_{fin}(*bü*) der Aktion *außerBetriebNehmen*(*bü*). Damit wird das Objekt *bü* entfernt, wenn ein Objekt der Klase *Verwaltung* diese Aktion ausführt.

Es gelten:

- $inBetriebNehmen(bü) \in \mathcal{E}_{Bahnübergang}$, $inBetriebNehmen(bü) \in \mathcal{A}_{Verwaltung}$ und
- $außerBetriebNehmen(bü) \in \mathcal{E}_{Bahnübergang}$, $außerBetriebNehmen(bü) \in \mathcal{A}_{Verwaltung}$.

5 Dynamisches Verhalten von Objekten

Das dynamische Verhalten von Objekten besteht darin, dass sie Aktionen ausführen, Zustände einnehmen bzw. auf die Ereignisse reagieren, die von den anderen Objekten des Objektsystems oder von der Umgebung ausgelöst werden. Dieses Verhalten wird hier lokales Verhalten eines Objekts genannt und mittels eines Zustandsdiagramms der UML spezifiziert. Die Teilnahme an Systemprozessen in Form von Interaktionen mit anderen Objekten des Objektsystems wird durch Sequenzdiagramme der UML beschrieben.

Mittlerweile wurden viele formale Semantiken für die UML-Notationen insbesondere für die Zustandsdiagramme der UML wie z. B. in [LMM99a, LQV01, LP99, VdB01 und HK04] definiert. Dabei wurden aber die softwarespezifischen Charakteristika dieser Diagramme in den Vordergrund gestellt, die für eine Implementierung von Softwareobjekten signifikant sind. In [DGH02, JW01, KMR01 und LMM99b] werden auch Formalisierungen für diese Diagramme vorgestellt, die eine Grundlage für die formale Verifikation bilden. Dabei werden jeweils einige Konstrukte, wie entry, do, exit, parallele Regionen, History-States etc. nicht behandelt.

In diesem Kapitel werden für die Zustandsdiagramme und die Sequenzdiagramme der UML und für die darin enthaltenen Notationen eindeutige und formale Definitionen eingeführt. Diese Definitionen ermöglichen nicht nur eine präzise Beschreibung des dynamischen Verhaltens der Objekte, sie bilden auch eine Basis für die Durchführung einer formalen Verifikation, die in Kapitel sieben diskutiert wird.

Für die Einführung dieser Definitionen werden die Begriffe Aktion, Aktionenfolge, Ereignis, Zustand und Zustandsüberführung genauer betrachtet. Dabei wird zusätzlich zu einer textuell formalen Beschreibung eine grafische Notation verwendet, die den Notationen in den Zustandsdiagrammen der UML [OMG03] entspricht. Eine umfassende und korrekte Darstellung aller diskutierten Fälle durch Grafiken ist nicht nur schwierig, sondern auch in manchen Fällen nicht möglich. Daher wird die grafische Notation lediglich zur Veranschaulichung bestimmter Fälle verwendet, die bei dem diskutierten Thema im Mittelpunkt stehen.

5.1 Aktionen und Aktionenfolgen

Eine Aktion ist die Ausführung einer Aktivität bzw. einer Handlung. Sie kann der Aufruf einer Funktion, das Schalten eines Schalters, das Anzeigen einer Meldung, das Berechnen bzw. die Übergabe eines Wertes, das Absenden einer Nachricht etc. sein. Ein Objekt kann in einer bestimmten Situation eine einzelne Aktion oder eine Sequenz von Aktionen Ausführen. Die Aktionen können aus weiteren Aktionen bestehen. Die Verfeinerung der Aktionen und ihre Darstellung in ihren einzelnen Bestandteilen hängen von der Sichtweise des Modellierers und von der gewünschten Verfeinerung in der Systemanforderungsspezifikation ab. Daher kann eine Handlung, die selbst aus mehreren Aktionen besteht, als eine einzige Aktion aufgefasst und erst in den weiteren Entwicklungsphasen verfeinert werden. Die Ausführung von Aktionen verbraucht Zeit. Folglich können die Aktionen unterbrochen oder vollständig ausgeführt werden.

In der vorliegenden Arbeit ist eine Aktion eine Handlung und eine Aktionenfolge ist eine Sequenz von Aktionen, die unmittelbar nacheinander ausgeführt werden können.

Aktionen bzw. Aktionenfolgen werden vollständig ausgeführt, sobald ihre Ausführungsbedingung erfüllt ist und solange sie nicht unterbrochen werden. Die Ausführungsbedingung von Aktionen sowie die Umstände für die vollständige Ausführung oder für die Unterbrechung einer Aktion bzw. einer Aktionenfolge werden in den nächsten Abschnitten erläutert.

5.1.1 Bezeichnung von Aktionen und Aktionenfolgen

Für die Bezeichnung der einzelnen Aktionen werden hier a_1 , a_2 , ... und für die Bezeichnung der Aktionenfolgen A_1 , A_2 , ... verwendet. Weiter wird a_ε leere Aktion genannt und bezeichnet die Situation, in der das Objekt keine Aktion ausführt. Die leere Aktion verbraucht keine Zeit.

5.1.2 Ausführung von Aktionen und Aktionenfolgen

Aktionen und Aktionenfolgen können sequenziell oder parallel bzw. zeitlich überlappend ausgeführt werden. Bei sequenzieller Ausführung kann jede Aktion erst nach dem Abschluss der vorherige Aktion ausgeführt werden, während bei paralleler Ausführung die Aktionen bzw. Aktionenfolgen zeitlich überlappend ausgeführt werden. Diese Fälle werden in den nächsten Abschnitten genauer erläutert.

5.1.2.1 Sequenzielle Ausführung von Aktionen und Aktionenfolgen

Eine sequenzielle Ausführung von Aktionen wird durch Kommata und eine sequenzielle Ausführung von Aktionenfolgen durch Semikolon gekennzeichnet.

So steht die Bezeichnung $A = a_1 , ... , a_n$ mit $n \in \mathbb{N}$ für die Aktionenfolge A, die aus einzelnen Aktionen a_1 bis a_n besteht. $A_1 ; ... ; A_m$ mit $m \in \mathbb{N}$ bezeichnet dann eine Folge von Aktionenfolgen, die sequenziell Ausgeführt werden.

Die Ausführung von $A = a_1 , ... , a_n$ mit $n \in \mathbb{N}$ beginnt mit der Ausführung von a_1 , wenn die Ausführungsbedingung von A und die von a_1 erfüllt sind. Nachdem die Aktion a_1 ausgeführt und abgeschlossen wurde, kann die Ausführung der darauf folgenden Aktion a_2 beginnen. Ist die Ausführungsbedingung von a_2 erfüllt, so wird a_2 ausgeführt und abgeschlossen.

Nach dem gleichen Prinzip werden alle Aktionen unmittelbar nacheinander ausgeführt und abgeschlossen, wenn nach dem Abschluss jeder Aktion die Ausführungsbedingung der darauf folgenden Aktion erfüllt ist. Die Ausführung von A wird mit dem Abschluss von a_n abgeschlossen, wenn nach dem Abschluss jeder Aktion a_i mit $1 \leq i < n$ die Ausführungsbedingung der nachfolgenden Aktion a_{i+1} erfüllt ist. In diesem Fall ist A vollständig ausgeführt. Ist die Ausführungsbedingung einer Aktion a_j mit $1 \leq j \leq n$ nicht erfüllt, so wird die Ausführung von A mit dem Abschluss von a_{j-1} abgeschlossen.

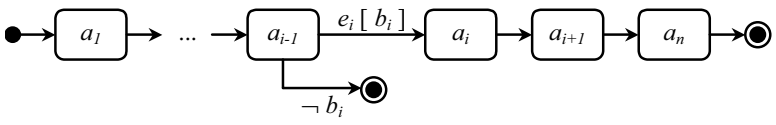


Abb. 5.1 Ausführung einer Aktionenfolge

Demnach ist es für die weiteren Betrachtungen wichtig, dass eine Aktionenfolge ausgeführt wird, wenn ihre Ausführungsbedingung erfüllt ist. Sie ist abgeschlossen, auch wenn die Ausführungsbedingung einer ihrer Aktionen nicht erfüllt ist und damit deren Ausführung an der entsprechenden Stelle abgebrochen ist. Das gilt auch für eine Aktionenfolge, deren Ausführungsbedingung erfüllt ist und die Ausführungsbedingung ihrer

ersten Aktion nicht erfüllt ist und dadurch keine ihrer Aktionen ausgeführt worden ist. Die grafische Darstellung der Ausführung einer Aktionenfolge ist in Abb. 5.1 gegeben.

Bei einer Folge von Aktionenfolgen A_1 ; ... ; A_n mit $n \in \mathbb{N}$ beginnt die Ausführung mit der Ausführung von A_1, wenn die Ausführungsbedingung von A_1 erfüllt ist. Nach dem Abschluss jeder Aktionenfolge, der auch erfolgt, wenn die Ausführung der dazugehörigen ersten Aktion nicht erfüllt ist und damit keine ihrer Aktionen ausgeführt wird, kann die darauf folgende Aktionenfolge ausgeführt werden, wenn ihre Ausführungsbedingung erfüllt ist. Somit endet die Ausführung von A_1 ; ... ; A_n mit dem Abschluss der Ausführung von A_n .

Durch Bildung einer Folge von Aktionenfolgen ist die Möglichkeit gegeben, die sequenzielle Ausführung von ereignisgesteuerten bzw. bedingten Aktionen unabhängig voneinander zu modellieren, denn auch wenn in A_1 ; ... ; A_n die Aktionenfolge A_{i-1} bei letzter ausführbarer Aktion und eventuell nicht vollständig endet, ist A_{i-1} abgeschlossen und damit kann dann mit der Ausführung von A_i begonnen werden. Demzufolge kann A_i ohne eine vollständige Ausführung aller Aktionen von A_{i-1} ausgeführt werden.

5.1.2.2 Parallele Ausführung von Aktionen und Aktionenfolgen

Die Bezeichnung a_1 ... a_n mit $n \in \mathbb{N}$, in der die Aktionen durch Leerzeichen getrennt sind, wird für eine parallele bzw. zeitlich überlappende Ausführung von Aktionen a_1 bis a_n verwendet. Dabei beginnen alle a_i mit $1 \leq i \leq n$ gleichzeitig, deren Ausführungsbedingungen erfüllt sind. Sie können jedoch zu unterschiedlichen Zeiten beendet werden.

Ebenso wird für eine zeitlich überlappende Ausführung von Aktionenfolgen A_1 bis A_n mit $n \in \mathbb{N}$ die Bezeichnung A_1 ... A_n verwendet. Dabei können zwar die ersten Aktionen aller Aktionenfolgen beim Gelten der jeweiligen Ausführungsbedingungen gleichzeitig beginnen, jedoch die entsprechenden darauf folgenden Aktionen müssen nicht gleichzeitig beginnen, weil die ersten Aktionen zu unterschiedlichen Zeiten beendet werden können, das heißt, während in einer Aktionenfolge die erste Aktion noch ausgeführt wird, ist in einer anderen Aktionenfolge die Ausführung der ersten Aktion beendet und die der darauf folgenden Aktion begonnen worden.

Bei $A_1\ A_2$ mit $A_1 = a_1$, a_2 und $A_2 = a_3$, a_4 , a_5 beginnt die Ausführung mit $a_1\ a_3$. Da jedoch a_1 (bzw. a_3) während der Ausführung von a_3 (bzw. a_1) beendet werden kann, ist die nächste mögliche Ausführung $a_2\ a_3$ (bzw. $a_1\ a_4$). Ebenso können a_1 und a_3 gleichzeitig beendet werden. In diesem Fall wird $a_2\ a_4$ ausgeführt. Abb. 5.2 zeigt die grafische Darstellung der möglichen Ausführungen bei $A_1\ A_2$.

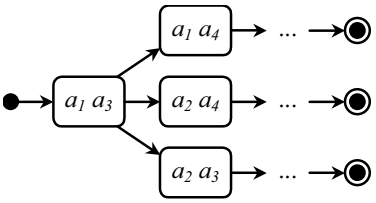


Abb. 5.2 Ausführung zweier paralleler Aktionenfolgen

5.1.3 Ausführungsbedingung von Aktionen und Aktionenfolgen

Damit eine Aktion ausgeführt werden kann, muss ihre Ausführungsbedingung erfüllt sein. Die Ausführung von a_i , die nach dem Abschluss der Ausführung einer Aktion a_j bzw. einer Aktionenfolge A_j erfolgen kann, ist von einem Ereignis e_i bzw. von einer Bedingung b_i abhängig. Die Kombination (e_i , b_i) ist die Ausführungsbedingung von a_i und ist erfüllt, wenn unmittelbar nach dem Abschluss von a_j bzw. nach dem Abschluss von A_j das Ereignis e_i eintritt und die Bedingung b_i erfüllt ist.

Es gelten $e_i = e_\varepsilon$ und $b_i = b_\varepsilon$, wenn bei der Modellierung für die Ausführung von a_i keine explizite Ausführungsbedingung definiert ist.

Ebenfalls lassen sich Ausführungsbedingungen auch für Aktionenfolgen definieren. Für eine Aktionenfolge $A = a_1 , ... , a_n$ mit $n \in \mathbb{N}$ ist die Kombination (e, b) die Ausführungsbedingung. Sie ist erfüllt, wenn unmittelbar vor der Ausführung von A das Ereignis e eintritt und die Bedingung b erfüllt ist.

Ist (e_1 , b_1) die Ausführungsbedingung von a_1 , gilt nicht generell (e, b) = (e_1 , b_1).

5.2 Ereignisse

Ein Ereignis ist eine Nachricht, eine Abfrage oder ein Befehl. Das dynamische Verhalten von Objekten wird u.a. von Ereignissen gesteuert. Ein Ereignis kann die Aufforderung zur Durchführung einer Aktion oder die Abfrage nach dem aktuellen Zustand des Objekts sein. Die Ereignisse werden von den anderen Objekten des Objektsystems oder von der Umgebung, welche die Nachricht oder die Abfrage senden, generiert. Dabei führen diese Objekte eine entsprechende Aktion aus. Für ein Objekt O sind daher Ereignisse die Aktionen anderer Objekte, die mit O kommunizieren.

Für die Bezeichnung von Ereignissen werden e_1 , e_2 , ... verwendet. Wenn die Ausführung einer Aktivität nicht von einem Ereignis abhängt, wird das leere Ereignis verwendet, das durch das Symbol e_ε gekennzeichnet wird. Das leere Ereignis wird als ständig vorhanden angenommen.

5.3 Zustand

Ein Zustand umfasst das dynamische Verhalten eines Objekts während eines bestimmten Lebensabschnitts. Zu Beginn, zum Schluss und während dieses Lebensabschnitts kann das Objekt eine Reihe von verschiedenen Aktivitäten bzw. gar keine Aktivität ausführen. Ein Zustand kann, wie folgt, beschrieben werden.

5.3.1 Definition (Zustand)

Für einen Zustand [z, $en(z)$, $act_1(z)$ ... $act_k(z)$, $do(z)$, $include(z)$, $ex(z)$] sind

- z der Zustandsname,
- $en(z)$ eine Aktionenfolge, die als *entry-actions* bezeichnet wird,
- $act_1(z)$ ... $act_k(z)$ mit $k \geq 0$ Aktionenfolgen,
- $do(z)$ eine Aktionenfolge, die als *do-actions* bezeichnet wird,
- $include(z)$ eine eventuell leere Menge weiterer Zustände und
- $ex(z)$ eine Aktionenfolge, die als *exit-actions* bezeichnet wird.

Dabei können die Aktionenfolgen $en(z)$, $act_1(z)$ ... $act_k(z)$, $do(z)$, $ex(z)$ teilweise oder alle aus der leeren Aktion a_ε bestehen.

5.3.1.1 Zustandsname

Der Zustandsname ist eine eindeutige Bezeichnung für den Zustand. Die Zustände eines Objekts haben paarweise unterschiedliche Namen.

5.3.1.2 entry-actions

Die entry-actions $en(z)$ sind die Aktionenfolgen, die beim Eintreten in den Zustand ausgeführt werden. Während der Ausführung dieser Aktionenfolgen kann der Zustand nicht verlassen werden, daher wird nicht auf die Austrittsbedingungen des Zustands (Abschnitt 5.3.6) reagiert. Die entry-actions eines Zustands z werden unmittelbar beim Eintritt in z ausgeführt.

5.3.1.3 Aktionenfolgen $act_1(z)$ ... $act_k(z)$

Die Aktionenfolgen $act_1(z)$ ... $act_k(z)$ ohne Konstrukte entry, do, oder exit sind Aktionenfolgen, die in z nur einmal und ohne Wiederholung ausgeführt werden. Die Ausführung dieser Aktionenfolgen beginnt nach dem Beenden der entry-actions.

Diese Aktionenfolgen werden laut UML ausgeführt, wenn das Objekt sich in dem Zustand z befindet. Das impliziert eine Ausführung von act_1 (z) ... act_k (z), sobald $en(z)$ beendet wurde, das heißt, sobald z aktiv ist. Da die einzelnen Aktionenfolgen in act_1 (z) ... act_k (z) voneinander unabhängig sind, kann für ihre Ausführungen keine bestimmte Reihenfolge festgelegt werden. Daher wird hier eine zeitlich überlappende Ausführung für diese Aktionenfolgen angenommen.

Während der Ausführung dieser Aktionenfolgen kann der Zustand verlassen werden, daher wird auf die Austrittsbedingungen des Zustands (Abschnitt 5.3.6) reagiert. In diesem Fall wird die Ausführung dieser Aktionen unterbrochen.

5.3.1.4 do-actions

Die do-actions $do(z)$ sind die Aktionenfolgen, die ausgeführt werden, solange sich das Objekt in z befindet. Die Ausführung dieser Aktionenfolgen beginnt unmittelbar nach dem Beenden der entry-actions und wird wiederholt, solange z nicht verlassen wurde. Wie bei Aktionenfolgen act_1 (z) ... act_k (z) kann z während der Ausführung der do-actions verlassen werden, es wird auf die Austrittsbedingungen von z (Abschnitt 5.3.6) reagiert. Dabei wird die Ausführung der do-actions unterbrochen.

Da für die Ausführungen von act_1 (z) ... act_k (z) und $do(z)$ keine Reihenfolge festgelegt werden kann, wird hier von einer zeitlich überlappenden Ausführung dieser Aktionenfolgen ausgegangen.

5.3.1.5 *include*(*z*)

Jeder Zustand kann aus weiteren Zuständen bestehen. Diese Zustände werden in textueller Notation unter *include* zusammengefasst. Die Zustände in include eines Zustandes z heißen direkte Teilzustände von z und z ist ihr direkter Oberzustand.

Die Menge der direkten Teilzustände eines Zustands z wird mit $include(z)$ bezeichnet.

Da jeder Teilzustand von z selbst weitere Teilzustände haben kann, wird unter den Teilzuständen eine Hierarchie gebildet. Damit hat der Zustand z weitere Teilzustände, die nicht seine direkten Teilzustände sind. Die Menge $in(z)$ ist dann die Menge aller Teilzustände von z und z ist ein Oberzustand für jedes Element von $in(z)$. Für $include(z)$ und $in(z)$ gilt $include(z) \subseteq in(z)$.

Die Aktivitäten der Teilzustände eines Zustands können entweder zeitlich überlappend stattfinden oder sie schließen sich gegenseitig aus. Für zwei direkte Teilzustände z_i und z_j eines Zustands z zeigt $z_i \parallel z_j$ den zeitlich überlappenden Ablauf der dazugehörigen Aktivitäten. In diesem Fall heißen z_i und z_j parallele Regionen von z. In Abb. 5.3 sind die Teilzustände z_1 , ... , z_n parallele Regionen von z.

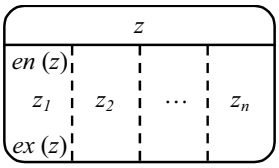


Abb. 5.3 Ein Zustand mit parallelen Teilzuständen

Sind z_i und z_j Zustände ohne einen Oberzustand oder sind sie direkte Teilzustände eines Zustands z , so zeigt die Bezeichnung $z_i \nparallel z_j$ den Ausschluss der gleichzeitigen Ausführung der Aktivitäten dieser Zustände. In diesem Fall heißen z_i und z_j nichtparallele Zustände. Abb. 5.4 zeigt z_1 , ... , z_n als Teilzustände von z mit $z_1 \nparallel ... \nparallel z_n$. In dieser Abbildung zeigen die Pfeile die Transitionen bzw. die Zustandswechsel, die in Abschnitt 5.4 erläutert werden.

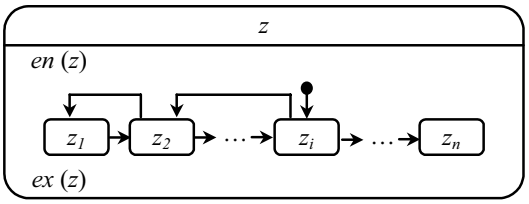


Abb. 5.4 Ein Zustand mit nichtparallelen Teilzuständen

Bezüglich der direkten Teilzustände z_1 , ... , z_n mit $n \in \mathbb{N}$ eines Zustands z gelten die folgenden Aussagen:

1. Aus $z_1 \parallel ... \parallel z_n$ folgen $en(z) = en(z_i)$ und $ex(z) = ex(z_i)$ für alle i mit $1 \leq i \leq n$.
2. Im Fall von $z_1 \nparallel ... \nparallel z_n$ wird ein z_i mit $1 \leq i \leq n$ Anfangszustand genannt und mit $default(z)$ bezeichnet. In Abb. 5.4 ist $z_i = default(z)$.

Mit *default*() wird der Anfangszustand von Zuständen ohne einen Oberzustand bezeichnet.

5.3.1.6 exit-actions

Die exit-actions $ex(z)$ sind die Aktionenfolgen, die ausgeführt werden, wenn der Zustand verlassen wird. Während der Ausführung dieser Aktionenfolgen wird nicht auf die Bedingungen reagiert, die weitere Aktivitäten des Zustands steuern. Daher werden diese Aktionenfolgen ohne Unterbrechung ausgeführt. Die Ausführung der exit-actions beginnt unmittelbar nach dem Gelten einer Austrittsbedingung des Zustands z.

In Abb. 5.5 ist in Teil a) ein Zustand mit seinen Aktivitäten dargestellt. Der Teil b) der Abbildung zeigt die Anordnung dieser Aktivitäten. Dabei zeigt *include*(z), dass die Ak-

tivitäten in den Teilzuständen parallel zu weiteren Aktivitäten des Zustands *z* ausgeführt werden.

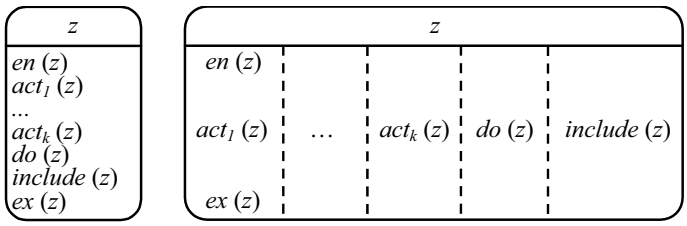


Abb. 5.5 die Anordnung der Aktivitäten in dem Zustand z

5.3.2 Aktiver Zustand

Ein Zustand *z* ist aktiv, sobald die Ausführung von *en*(*z*) beginnt und solange *ex*(*z*) nicht abgeschlossen ist.

Ist der Zustand *z* nicht aktiv, wird mit der Ausführung von *en*(*z*) begonnen, wenn eine Transition mit dem Eintritt in *z* oder in einen seiner Teilzustände stattfindet. Transitionen werden in Abschnitt 5.4 und die Reihenfolge der Ausführung beteiligter entry-actions wird in Abschnitt 5.3.4 erläutert. Da die entry-actions der Oberzustände eines Zustands *z* vor *en*(*z*) ausgeführt werden, sind alle Oberzustände von *z* aktiv, wenn *z* aktiv ist.

Ein aktiver Zustand, dessen entry-actions abgeschlossen sind, wird verlassen, sobald die Bedingung für sein Verlassen erfüllt ist. Nachdem diese Bedingung erfüllt wurde, wird *z* durch Ausführung von *ex*(*z*) verlassen. Mit dem Abschluss von *ex*(*z*) ist *z* nicht mehr aktiv. Die Bedingungen zum Verlassen des Zustands werden Austrittsbedingungen genannt und werden in Abschnitt 5.3.6 erläutert.

Für die Zustände z_1 , ... , z_n und $n \in \mathbb{N}$ gelten die folgenden Aussagen:

1. Im Fall von $z_1 \nparallel ... \nparallel z_n$ kann zu jedem Zeitpunkt nur ein z_i mit $1 \leq i \leq n$ aktiv sein.
2. Bei $z_1 \parallel ... \parallel z_n$ sind zu jedem Zeitpunkt entweder alle oder kein z_i mit $1 \leq i \leq n$ aktiv.

In den nächsten Abschnitten wird eine Verletzung dieser Aussagen mehrdeutiges bzw. ungültiges Verhalten genannt.

5.3.3 Aktivierung eines Zustands

Aktivierung eines Zustands *z* bedeutet die Ausführung seiner entry-actions *en*(*z*).

Da die Aktivitäten von z nur durchgeführt werden können, während z aktiv ist, werden die weiteren Aktivitäten von z, wie $do(z)$, $act_1(z)$ ... $act_k(z)$ und die Aktivitäten in den Teilzuständen, erst nach dem Abschluss von entry-actions $en(z)$ durchgeführt. Daher wird vorausgesetzt, dass während der Ausführung von $en(z)$ nicht auf die Austrittsbedingungen von z (Abschnitt 5.3.6) reagiert wird. Damit wird $en(z)$ ohne Unterbrechung ausgeführt und abgeschlossen.

Da die Aktivitäten in den Teilzuständen eines Zustands erst nach dem Abschluss von entry-actions des Zustands durchgeführt werden, folgt aus der Aktivierung eines Zustands z_i mit $z_i \in include(z)$, dass z aktiv ist. Daraus folgt, dass alle Oberzustände eines aktiven Zustands aktiv sind.

Wenn der Zustand z mit direkten Teilzuständen z_1 , ... , z_n und $n \in \mathbb{N}$ aktiviert wird, so kann im Fall von $z_1 \nparallel ... \nparallel z_n$ nach dem Abschluss von $en(z)$ der als Anfangszustand gekennzeichnete Teilzustand $default(z)$ aktiviert werden, wenn die Aktivierung von z nicht über einen seiner Teilzustände bzw. nicht über den History-State (Abschnitt 5.6.3) erfolgt. Diese Fälle werden in den nächsten Abschnitten behandelt. Die Aktivierung von $default(z)$ erfolgt, sobald die Defaulttransition (Abschnitt 5.4.4) in $default(z)$ stattfinden kann.

Da im Fall von $z_1 \parallel ... \parallel z_n$ noch $en(z) = en(z_i)$ für alle z_i mit $1 \le i \le n$ gilt, werden alle Teilzustände gleichzeitig aktiviert, wenn $en(z)$ ausgeführt wird. Dabei wird die Folge $en(z)$; ($en(default(z_1))$ ... $en(default(z_n))$) für alle z_i ausgeführt, deren Defaulttransitionen stattfinden können.

5.3.4 Aktivierung eines Zustands über seine Teilzustände

Da die Aktivierung eines Zustands z_i ohne die Aktivierung seiner Oberzustände nicht möglich ist, wird unmittelbar vor der Aktivierung von z_i sein eventuell nicht aktiver direkter Oberzustand aktiviert. Daraus ergibt sich eine Aktivierung aller Oberzustände von z_i , die nicht aktiv sind. Formal lässt sich die Aktivierung eines Zustands, wie folgt, beschreiben:

Es seien Zustände z_1 , ... , z_n mit $n > 1$ und $z_1 \in include(z_2)$, ... , $z_{n-1} \in include(z_n)$ gegeben. Dabei seien z_{m+1} mit $m < n$ aktiv und z_m nicht aktiv. Bei der Aktivierung des Zustands z_i mit $i < m$ wird die Folge $en(z_m)$; $en(z_{m-1})$; ... ; $en(z_{i+1})$; $en(z_i)$ ausgeführt. Dabei werden alle beteiligten Oberzustände aktiv und können ihre weiteren definierten Aktivitäten durchführen.

Die Aktivierung von Zuständen wird in Abb. 5.6 gezeigt. Wenn z_6 aktiv ist, so sind die Oberzustände z_4 und z_5 aktiv jedoch der Zustand z_3 ist wegen $z_6 \nparallel z_3$ nicht aktiv. Wenn

nun nach z_6 der Zustand z_1 aktiviert wird, so werden die Zustände z_3 und z_2 über ihren Teilzustand z_1 aktiv und die Folge $en(z_3)$; $en(z_2)$; $en(z_1)$ wird ausgeführt.

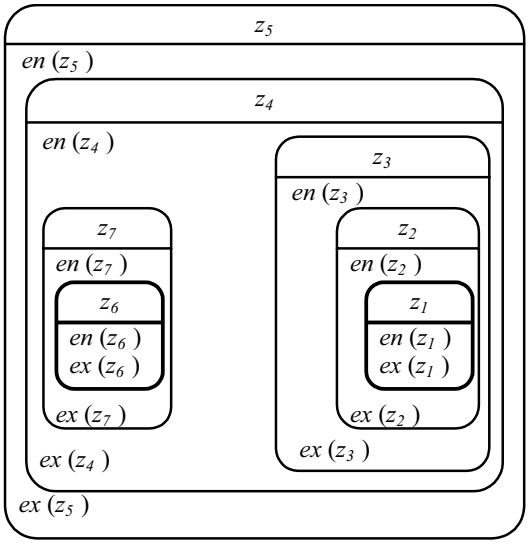


Abb. 5.6 Aktivierung von Zuständen

Ist einer der Zustände ein direkter Teilzustand eines Zustands mit parallelen Regionen, so werden bei Aktivierung dieses Zustands die als Anfangszustand gekennzeichneten Teilzustände der benachbarten Regionen aktiviert, deren Defaulttransitionen stattfinden können. Wenn unter gegebenen Voraussetzungen ein z_j mit $i \le j \le m$ ein direkter Teilzustand von z_{j+1} mit direkten Teilzuständen z'_1 , ... , z'_k und $z'_1 \parallel ... \parallel z_j \parallel ... \parallel z'_k$ ist, wird in der genannten Folge der Term $en(z_j)$ durch $(en(default(z'_1)) \; ... \; en(default(z'_k)))$ für alle z'_p mit $1 \le p \le k$ und $p \ne j$ ersetzt, deren Defaulttransitionen stattfinden können. Die Aktivierung von z_j erfolgt wegen $en(z_j) = en(z_{j+1})$ durch die Ausführung von $en(z_{j+1})$.

Dieser Fall wird in Abb. 5.7 veranschaulicht. Wenn z_6 aktiv ist, so sind die Oberzustände z_4 und z_5 aktiv und der Zustand z_3 ist wegen $z_6 \nparallel z_3$ nicht aktiv. Wenn nach z_6 der Zustand z_1 aktiviert wird, so werden die Zustände z_3 und z_2 über ihren Teilzustand z_1 aktiv. Die Aktivierung von z_2 hat zur Folge, dass alle zu z_2 parallelen Zustände z'_1 , ... , z'_k aktiviert werden. Dabei kann bei jedem Zustand $z'_i \ne z_2$ mit $1 \le i \le k$ der Teilzustand $default(z'_i)$ aktiviert werden, wenn seine Defaulttransition stattfinden kann. Dabei wird die Folge $en(z_3)$; $(en(default(z'_1)) \; ... \; en(z_1) \; ... \; en(default(z'_k)))$ ausgeführt. Aufgrund von $en(z_3) = en(z_2)$ kommt in dieser Folge $en(z_2)$ nicht vor. Ebenso $en(default(z_2))$ wird nicht ausgeführt, weil die Aktivierung von z_2 über den Teilzustand z_1 erfolgt.

Die Aktionenfolgen $en(default(z'_1))$ bis $en(default(z'_k))$ können unabhängig voneinander und zeitlich überlappend ausgeführt werden, sobald ihre Defaulttransitionen stattfinden können.

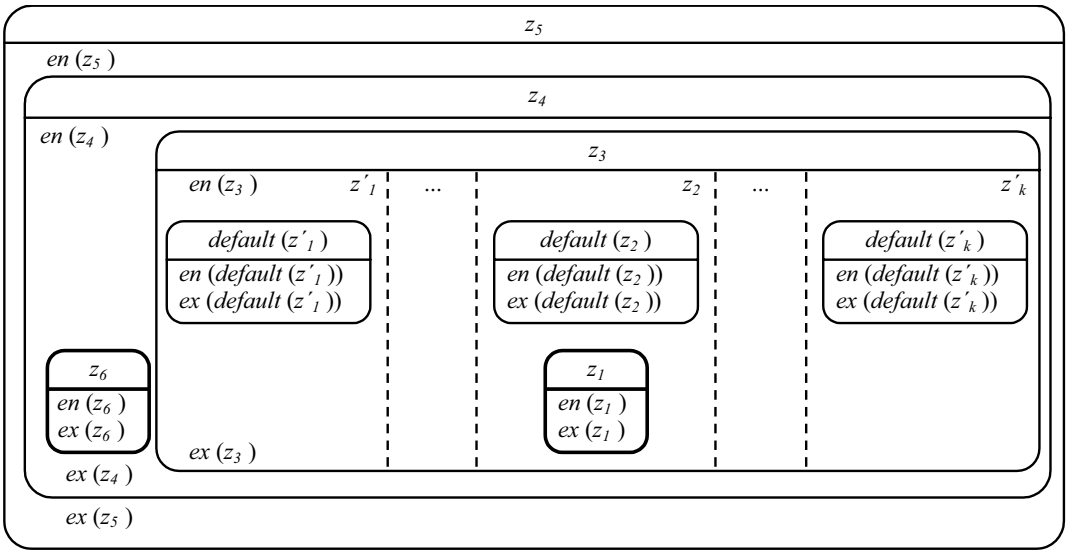


Abb. 5.7 Aktivierung von Zuständen mit parallelen Regionen

5.3.5 Verlassen eines Zustands

Ein aktiver Zustand bleibt solange aktiv, bis eine seiner Austrittsbedingungen (Abschnitt 5.3.6) erfüllt wird. Solange keine Austrittsbedingung erfüllt ist, wird der Zustand nicht verlassen, auch wenn er seine Aktivitäten beendet hat.

Das Verlassen eines aktiven Zustands z bedeutet den Abschluss der Ausführung seiner exit-actions $ex(z)$. Sobald für den aktiven Zustand z mit der Austrittsbedingung (e, b) das Ereignis e eintritt und die Bedingung b erfüllt ist, kann z verlassen werden. Beim Verlassen von z werden alle Aktivitäten von z, wie $do(z)$, $act_1(z)$ ... $act_k(z)$ und die Aktivitäten in den Teilzuständen unterbrochen. Daher wird während der Ausführung von $ex(z)$ nicht mehr auf die Ereignisse und Bedingungen reagiert, die die Aktivitäten in z steuern. Demzufolge wird $ex(z)$ ohne Unterbrechung ausgeführt und abgeschlossen.

Wenn ein Zustand z direkte Teilzustände z_1 , ... , z_n mit $n \in \mathbb{N}$ hat, so wird beim Verlassen von z der aktive Teilzustand z_i mit $1 \le i \le n$ verlassen. Hat z_i selbst Teilzustände, so führt das zu einem rekursiven Verlassen des aktiven direkten Teilzustands in jeder Hierarchieebene von Innen nach Außen. Das Verlassen von z und das von z_i erfolgen durch die Ausführungen ihrer exit-actions. Im Allgemeinen finden diese Ausführungen, wie folgt, statt.

5.3.5.1 Verlassen eines Zustands und seiner Teilzustände

Es seien z_1 , ... , z_n mit $n \in \mathbb{N}$ die direkten Teilzustände eines Zustands z. Weiter seien z und z_i mit $1 \le i \le n$ aktiv, (e, b) eine Austrittsbedingung von z und (e_i , b_i) eine Austrittsbedingung von z_i . Wenn e eintritt und b erfüllt ist, wird z verlassen und die Folge

- $ex(z_i)$; $ex(z)$ ausgeführt, wenn $z_1 \nparallel ... \nparallel z_n$ gilt, denn (e, b) ist auch eine Austrittsbedingung von z_i .
- $ex(z)$ ausgeführt, wenn $z_1 \parallel ... \parallel z_n$ gilt, denn (e, b) ist eine Austrittsbedingung von z_i und es gilt $ex(z_i) = ex(z)$.

Aus der Ausführung von $ex(z_i)$; $ex(z)$ bei $z_1 \nparallel ... \nparallel z_n$ folgt, dass im Fall von $z_1 \parallel ... \parallel z_n$ unmittelbar vor der Ausführung von $ex(z)$ die Ausführung $ex(H(z_1)) ... ex(H(z_n))$ stattfindet, wenn in jedem z_i der Zustand $H(z_i)$ der aktive Teilzustand ist. $H(z_i)$ ist der History-State von z_i und wird in Abschnitt 5.6.3 erläutert. In dieser Ausführung wird für alle z_i mit $1 \leq i \leq n$ der Term $ex(H(z_i))$ durch a_ε ersetzt, wenn vor dem Gelten von (e, b) kein Teilzustand von z_i aktiv war.

In der Abb. 5.7 ist dieser Fall beim Zustand z_4 gegeben. Wenn der Zustand z_1 aktiv ist und z_4 verlassen wird, wird die Folge $ex(z_1)$; $ex(z_3)$; $ex(z_4)$ ausgeführt. In dieser Folge kommen $ex(z'_1)$, ... , $ex(z'_k)$ und $ex(z_2)$ nicht vor, denn aufgrund von $z'_1 \parallel ... \parallel z_2 \parallel ... \parallel z_k$ diese Aktionenfolgen mit $ex(z_3)$ identisch sind.

5.3.6 Austrittsbedingungen eines Zustands

Für das Verlassen eines aktiven Zustands muss eine seiner Austrittsbedingungen erfüllt sein. Die Austrittsbedingungen werden im Kontext von Transitionen angegeben, die in Abschnitt 5.4 erläutert werden. Eine Austrittsbedingung kann das Eintreten eines Ereignisses, das Beenden bestimmter Aktivitäten oder die Geltung einer Bedingung sein. Sie wird in Form von (e, b) notiert, wobei e ein (eventuell leeres) Ereignis und b eine (eventuell leere) Bedingung sind.

Wenn z mit der Austrittsbedingung (e, b) aktiv ist, so kann z verlassen werden, wenn e eintritt und b wahr ist.

Da die Aktivierung eines Zustands ohne die Aktivierung seiner Oberzustände nicht möglich ist und da bei dem Verlassen eines Zustands alle seine aktiven Teilzustände verlassen werden, gelten die folgenden Aussagen:

1. Der Eintritt in einen Zustand ist zugleich ein Eintritt in alle seine nicht aktiven Oberzustände.
2. Die Austrittsbedingung für einen Zustand ist zugleich eine Austrittsbedingung für alle seine Teilzustände, denn beim Verlassen des Zustands werden seine Teilzustände verlassen.

Um ein ungültiges bzw. ungültiges Verhalten des Objekts zu vermeiden, unterliegen die Austrittsbedingungen eines Zustands bestimmten Regeln, die in Abschnitt 5.5.3 disku-

tiert werden. Hier werden zunächst die Auswirkung der Austrittsbedingungen und ihre Rolle als Ursache für einen Zustandswechsel erläutert.

5.4 Transitionen

Die Überführung von Zuständen ineinander erfolgt durch Transitionen. Eine Transition ist die Verknüpfung zwischen dem Verlassen eines Zustands und dem Eintritt in den Folgezustand. Dabei wird die Transition von einer Austrittsbedingung eines Zustands gesteuert. Wenn der Zustand aktiv und seine Austrittsbedingung erfüllt ist, findet die Transition statt. Sie veranlasst dann das Verlassen des Zustands und den Eintritt in den Folgezustand.

5.4.1 Definition (Transition)

> Eine Transition $t = z_i \xrightarrow{e[b]/A} z_j$ ist der Zustandswechsel von z_i mit einer Austrittsbedingung (e, b) nach z_j . Der Ausdruck e [b] / A wird die Beschriftung von t genannt. Dieser Zustandswechsel kann stattfinden, wenn z_i aktiv ist, das Ereignis e eintritt und die Bedingung b gilt, das heißt, wenn die Austrittsbedingung von z_i erfüllt ist.

Wenn diese Transition stattfindet, werden z_i verlassen, die Aktionenfolge A ausgeführt und z_j aktiviert. In UML [OMG03] wird t in Bezug auf z_i eine ausgehende Transition genannt.

Bei einer Transition von z_i nach z_j finden nicht nur das Verlassen von z_i und der Eintritt in z_j statt, sondern alle Oberzustände von z_i werden verlassen, die keine Oberzustände von z_j sind und alle Oberzustände von z_j werden aktiviert, die keine Oberzustände von z_i sind. Formal heißt das:

5.4.2 Ausführung von Aktionenfolgen infolge einer Transition

Bei der Transition $t = z_i \xrightarrow{e[b]/A} z_j$ mit $z_i \in in(z_n)$ und $z_j \in in(z_m)$ werden alle Zustände z_p mit $z_i \in in(z_p)$, $z_j \notin in(z_p)$ verlassen und alle Zustände z_q mit $z_j \in in(z_q)$, $z_i \notin in(z_q)$ aktiviert. Die Ausführung der beteiligten Aktionenfolgen erfolgt unmittelbar in der folgenden Reihenfolge:

1. Die exit-actions aller Oberzustände von z_i , die keine Oberzustände von z_j sind, gemäß in Abschnitt 5.3.5.1 gezeigtem Schema,
2. Die Ausführung der Aktionenfolge A und

3. Die entry-actions aller Oberzustände von z_j , die keine Oberzustände von z_i sind, gemäß in Abschnitt 5.3.4 gezeigtem Schema.

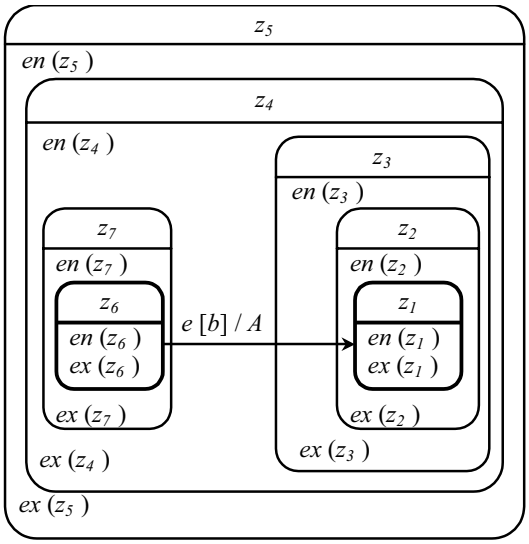


Abb. 5.8 Transition

Wenn in Abb. 5.8 der Zustand z_6 aktiv ist und die Austrittsbedingung (e, b) gilt, dann findet die Transition $z_6 \xrightarrow{e[b]/A} z_1$ statt. Dabei werden die Zustände z_6 und z_7 verlassen und ihre exit-actions werden ausgeführt, denn z_6 und z_7 sind keine Oberzustände von z_1 . Die Zustände z_3 , z_2 und z_1 werden aktiviert und ihre entry-actions werden ausgeführt, denn sie sind keine Oberzustände von z_6 .

Daher wird die Folge $ex(z_6)$; $ex(z_7)$; A; $en(z_3)$; $en(z_2)$; $en(z_1)$ ausgeführt, wenn die Transition $z_6 \xrightarrow{e[b]/A} z_1$ stattfindet.

Obwohl in UML [OMG03] Transitionen zwischen Teilzuständen zweier paralleler Regionen (Abb. 5.9) nicht untersagt sind, werden hier derartige Transitionen nicht zugelassen. Diese Entscheidung basiert auf dem Folgenden:

Wenn in einem Zustand z mit parallelen Teilzuständen eine Transition von einem Teilzustand einer Region z_i in einen der Teilzustände der Region z_j stattfinden würde, so würde z_i verlassen, während z nicht verlassen worden ist. Das hat zur Folge, dass z noch aktiv ist und dabei sich das Objekt in keinem der definierten Teilzustände von z_i befindet.

Um diese Anomalie zu verhindern, wird durch die folgende Regel verlangt, dass zwischen Teilzuständen zweier paralleler Regionen keine Transitionen definiert werden.

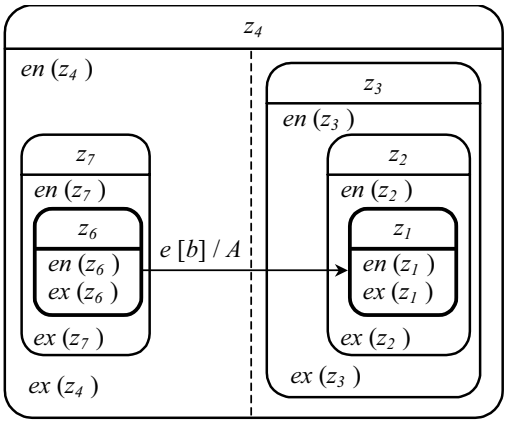


Abb. 5.9 Ungültige Transition zwischen parallelen Regionen

5.4.3 Regel

> Eine Transition $t = z_i \xrightarrow{e[b]/A} z_j$ mit $z_i \in in(z_n)$ und $z_j \in in(z_m)$ und $z_n \parallel z_m$ ist ungültig.

Eine derartige Transition ist in Abb. 5.9 durch $z_6 \xrightarrow{e[b]/A} z_1$ dargestellt. Diese Transition ist wegen $z_6 \in in(z_7)$, $z_1 \in in(z_3)$ und $z_7 \parallel z_3$ ungültig.

Eine formale Begründung dieser Regel ist, wie folgt,:

Für z_n und z_m gibt es einen Zustand z mit z_n , $z_m \in include(z)$. Da z_n und z_m parallele Regionen sind, gelten $z_i \notin in(z_m)$ und $z_j \notin in(z_n)$. Durch die Transition t werden z_i und z_n verlassen, das heißt, $ex(z_n)$ wird ausgeführt. Die Ausführung von $ex(z_n)$ hat zur Folge, dass z verlassen wird, denn es gilt $ex(z_n) = ex(z)$. Der Zustand z darf jedoch nicht verlassen werden, weil $z_j \in include(z)$ gilt.

Damit z nicht verlassen wird, kann $ex(z)$ nicht ausgeführt werden. Folglich kann der Zustand z_n nicht verlassen werden, denn es gilt $ex(z_n) = ex(z)$. Damit bleibt z_n aktiv. Daraus folgt, dass nach dem Verlassen von z_i der Folgezustand im bereits aktiven Zustand z_n nicht definiert ist.

5.4.4 Defaulttransition

Um bei nichtparallelen Zuständen den Entritt in den Anfangszustand zu definieren, wird eine Defaulttransition (Abb. 5.4) verwendet.

Wie bei anderen Transitionen kann eine Defaulttransition von einem Ereignis oder einer Bedingung gesteuert werden. Wenn eine Defaulttransition stattfindet, kann auch eine Aktionenfolge ausgeführt werden. Die Notation $\xrightarrow{e[b]/A}$ *default*(z) zeigt eine De-

faulttransition mit dem Ereignis *e*, der Bedingung *b* und der Aktionenfolge *A* für den Eintritt in den Anfangszustand *default*(*z*). Dabei können $e = e_\varepsilon$, $b = b_\varepsilon$ und $A = a_\varepsilon$ gelten.

Da eine Defaulttransition $\xrightarrow{e[b]/A}$ *default*(*z*) nicht infolge des Verlassens eines Zustands stattfindet, wird dabei in dem ersten Schritt des in Abschnitt 5.4.2 genannten Schemas die leere Aktion a_ε ausgeführt.

In der grafischen Notation wird die Defaulttransition durch einen Pfeil gekennzeichnet, der das Startsymbol (Abschnitt 5.6.1) mit dem Anfangszustand verbindet.

5.4.5 Initialtransition

Eine Initialtransition ist eine Defaulttransition $\xrightarrow{e[b]/A}$ *default*(), bei der $e = e_{init}$ und *default*() der erste Zustand ist, in dem sich das Objekt nach dem Erzeugen befindet. Dieser Zustand ist der Anfangszustand, der keinen Oberzustand hat.

5.4.6 Finaltransition

Bei nichtparallelen Zuständen geschieht der Übergang in den Endzustand (Abschnitt 5.6.2) durch eine Finaltransition.

Eine Finaltransition hat die Form $z_i \xrightarrow{e[b]/A} final()$ bzw. $z_i \xrightarrow{e[b]/A} final(z)$. Bei einer Finaltransition der Form $z_i \xrightarrow{e[b]/A} final()$ gilt $e = e_{fin}$.

5.5 Prioritätsregeln für Austritts- bzw. Ausführungsbedingungen

Die Ereignisse bzw. die Bedingungen, die in Zusammenhang mit einer Austrittsbedingung eines Zustands definiert werden, können unter Umständen gleichzeitig mit den Ereignissen eintreten bzw. mit den Bedingungen erfüllt werden, die für weitere Austrittsbedingungen des Zustands definiert sind. Ebenfalls können Austrittsbedingungen eines Zustands gleichzeitig mit den Ausführungsbedingungen erfüllt werden, die Aktionen bzw. Aktionenfolgen des Zustands steuern.

Für die Bezeichnung dieser Fälle wird hier bezüglich der Austritts- bzw. Ausführungsbedingungen der Begriff konkurrierend, wie folgt, eingeführt:

Zwei Austrittsbedingungen eines Zustands oder Ausführungsbedingungen von Aktionen und Aktionenfolgen sind *konkurrierend*, wenn die Möglichkeit besteht, dass ihre Ereignisse gleichzeitig eintreten und ihre Bedingungen gleichzeitig wahr sind.

Eine Formale Beschreibung dieser Begriffe lautet:

5.5.1 Definition (konkurrierende Bedingungen)

Es sei z ein Zustand und (e_i , b_i) und (e_j , b_j) Austrittsbedingungen von z bzw. Ausführungsbedingung von Aktionen oder Aktionenfolgen in z. Wenn die Möglichkeit besteht, dass e_i und e_j gleichzeitig eintreten und b_i und b_j gleichzeitig wahr sind, dann sind (e_i , b_i) und (e_j , b_j) *konkurrierend.*

Wenn zwei Austrittsbedingungen eines Zustands konkurrierend sind, so kann beim gleichzeitigen Erfüllen dieser Austrittsbedingungen ein ungültiges Verhalten des Objekts vorkommen. Wenn derartige Austrittsbedingungen Transitionen steuern, die nichtparallele Zustände aktivieren, so ist der Eintritt in den Folgezustand undefiniert.

Ebenso können eine Austrittsbedingung eines Zustands und die Ausführungsbedingung einer seiner Aktionen konkurrierend sein. In diesem Fall muss bestimmt werden, ob die Austrittsbedingung den Vorrang hat, das heißt, ob der Zustand verlassen wird und die Aktion nicht ausgeführt wird, oder die Ausführungsbedingung den Vorrang hat, das heißt, ob die Aktion ausgeführt wird und der Zustand nicht verlassen wird.

In Bezug auf die Austrittsbedingungen eines Zustands und die Ausführungsbedingungen seiner internen Aktionen ist in UML [OMG03] durch die intuitive Semantik der Austrittsbedingung des Zustands eine höhere Priorität gegeben. Damit werden alle internen Aktivitäten des Zustands unterbrochen und der Zustand wird verlassen, sobald die Austrittsbedingung erfüllt ist. Diese Prioritätsvergabe hat die folgenden Unklarheiten:

1. Zu internen Aktivitäten eines Zustands gehören unter anderem die entry-actions, deren Ausführung zur Aktivierung des Zustands führt. Sind eine Austrittsbedingung des Zustands und die Ausführungsbedingung einer Aktion von entry-actions konkurrierend, so kann nach dieser Priorität der Zustand vor der Ausführung dieser Aktion verlassen werden, wenn die Ausführungsbedingung dieser Aktion zur gleichen Zeit mit der Austrittsbedingung des Zustands erfüllt werden. Das bedeutet, dass der Zustand verlassen wird, bevor er durch Ausführung von entry-actions überhaupt aktiviert wurde.
2. Da die Angabe eines Ereignisses, einer Bedingung und einer Aktion im Kontext einer Transition optional ist, sind Austrittsbedingungen ohne ein Ereignis und ohne eine Bedingung erlaubt. Durch die Priorität der Austrittsbedingung wird ein Zustand mit derartiger Austrittsbedingung sofort verlassen und seine internen Aktionen können nie zur Ausführung kommen, denn die Austrittsbedingung mit leerem Ereignis und leerer Bedingung ist immer erfüllt und veranlasst den Austritt aus dem Zustand.

Darüber hinaus können die Austrittsbedingungen eines Zustands und die seiner Teilzustände konkurrierend sein. Das führt dazu, dass bei dem gleichzeitigen Erfüllen dieser Austrittsbedingungen der Folgezustand nicht eindeutig bestimmt ist. Dieser Fall ist in UML [OMG03] dadurch gelöst, dass der Austrittsbedingung des Teilzustands eine hö-

here Priorität gegeben ist. In diesem Fall wird der Teilzustand verlassen und der Oberzustand bleibt aktiv, indem der Folgezustand innerhalb des Oberzustands aktiviert wird. Bei klassischen State-Charts [Har87] haben die Austrittsbedingungen des Oberzustands eine höhere Priorität. Demzufolge werden der Oberzustand und alle seine aktiven Teilzustände verlassen.

Im Gegensatz zu Softwaresystemen, in denen für alle Objekte des Systems das gleiche Verhaltensmuster festgesetzt werden kann, lassen sich die realen Objekte wie die Komponenten eines Eisenbahnsystems nicht durch ein einheitliches Verhaltensschema charakterisieren. Aus diesem Grund werden in der vorliegenden Arbeit für die Austrittsbedingungen eines Zustands und die seiner Teilzustände keine generellen Prioritäten eingeräumt. Eine generelle Priorität für die eine oder die andere Austrittsbedingung verursacht eine Einschränkung bei der Modellierung des dynamischen Verhaltens realer Objekte.

Um in diesen Fällen ein mehrdeutiges Verhalten der Objekte zu vermeiden, wird hier bezüglich eines Zustands und seiner Teilzustände ein gleichzeitiges Gelten der Austrittsbedingungen generell nicht zugelassen bzw. nur in bestimmten Fällen erlaubt.

Bei einem gleichzeitigen Gelten von Austrittsbedingungen eines Zustands und Ausführungsbedingungen seiner Aktionen bzw. Aktionenfolgen wird durch die Einführung von Prioritätsregeln die in der UML [OMG03] erwähnte Priorität weiter präzisiert.

Die folgenden Regeln klären das Verhältnis zwischen den Austrittsbedingungen eines Zustands und Ausführungsbedingungen seiner Aktionen bzw. Aktionenfolgen:

5.5.2 Regel

> Konkurrierende Austrittsbedingungen dürfen nur dann für einen Zustand definiert werden, wenn sie Transitionen mit Eintritt in den gleichen Zustand oder in Zustände steuern, deren Oberzustände parallele Regionen eines dritten Zustands sind.

Formal lässt sich diese Forderung, wie folgt, formulieren:

Die Austrittsbedingungen (e_i , b_i) und (e_j , b_j) eines Zustands z sind nur dann konkurrierend, wenn (e_i , b_i) und (e_j , b_j) Transitionen t_i und t_j mit Aktionenfolgen A_i und A_j und $A_i = A_j$ in die Zustände z_i und z_j steuern, für die eine der folgenden Eigenschaften zutrifft:

- $z_i = z_j$ (Abb. 5.10).
- Es existieren z'_i , z'_j mit $z'_i \neq z'_j$, so dass $z_i \in in(z'_i)$, $z_j \in in(z'_j)$ und $z'_i \parallel z'_j$ gelten (Abb. 5.11).

Dabei wird A_i nur einmal ausgeführt.

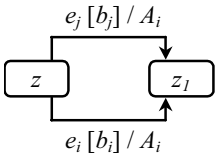


Abb. 5.10 Aktivierung eines Zustands über konkurrierende Austrittsbedingungen

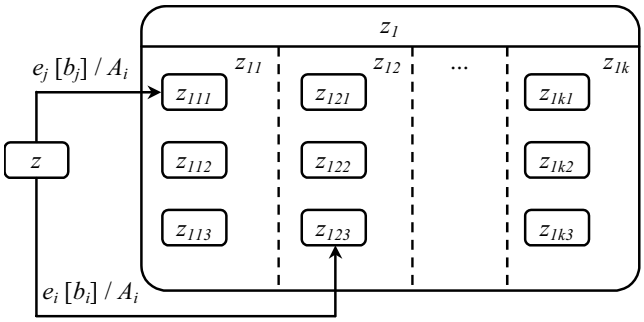


Abb. 5.11 Aktivierung paralleler Zustände über konkurrierende Austrittsbedingungen

Diese Regel lässt für einen Zustand die Existenz konkurrierender Austrittsbedingungen nur in dem Fall zu, wenn die Folgezustände bei beiden Austritten gleich sind oder wenn dadurch Teilzustände paralleler Regionen aktiviert werden. In den anderen Fällen sind die konkurrierenden Austrittsbedingungen für einen Zustand nicht erlaubt, denn dadurch kann die Aktivierung nichtparalleler Zustände und damit ein ungültiges bzw. ein mehrdeutiges Verhalten entstehen, wie es in 5.3.2 genannt wurde.

Die Anwendung dieser Regel auf Zustände mit Teilzuständen führt zu der Folgerung, dass die Austrittsbedingungen des Oberzustands und die der Teilzustände, welche die Aktivierung der weiteren Teilzustände veranlassen, nicht konkurrierend sein dürfen. Anderenfalls besitzt der Teilzustand konkurrierende Austrittsbedingungen, weil gemäß Abschnitt 5.3.6 die Austrittsbedingung des Oberzustands zugleich eine Austrittsbedingung aller Teilzustände ist.

5.5.3 Regel

Unter den Ausführungsbedingungen von Aktionen bzw. Aktionenfolgen eines Zustands und seiner Austrittsbedingung ohne leeres Ereignis und ohne leere Bedingung haben die Ausführungsbedingungen von entry-actions die höchste Priorität. Die Ausführungsbedingungen anderer Aktionen haben eine niedrigere Priorität als die Austrittsbedingung, die Austrittsbedingung mit leerem Ereignis und leerer Bedingung hat die niedrigste Priorität. Weiter darf für einen Zustand mit do-actions bzw. mit Teilzuständen ohne einen Endzustand keine Austrittsbedingung mit leerem Ereignis und leerer Bedingung definiert werden, denn sonst kann der Zustand

nicht verlassen werden, weil die do-actions bzw. die Aktivitäten der Teilzustände ständig wiederholt werden.

Eine formale Beschreibung dieser Regel lautet:

Es sei z ein Zustand mit einer Austrittsbedingung (e, b). Weiter sei $a_i \neq a_\varepsilon$ eine Aktion mit der Ausführungsbedingung (e_i , b_i), die in einer Aktionenfolge A in z ausgeführt wird. Dann gelten bezüglich (e, b) und (e_i , b_i) die folgenden Aussagen:

1. (e_i , b_i) hat eine höhere Priorität, wenn $A = en(z)$ gilt.
2. (e_i , b_i) hat eine höhere Priorität, wenn (e, b) = (e_ε , b_ε).
3. (e, b) hat eine höhere Priorität, wenn $A \neq en(z)$ und (e, b) $\neq$ (e_ε , b_ε) gilt.
4. Wenn (e, b) = (e_ε , b_ε) gilt, dann gelten $do(z) = a_\varepsilon$ und $include(z) = \varnothing$ oder $final(z) \in include(z)$ bzw. aus $do(z) \neq a_\varepsilon$ oder $include(z) \neq \varnothing$ und $final(z) \notin include(z)$ folgt (e, b) $\neq$ (e_ε , b_ε).

Durch den ersten Teil dieser Regel wird der Ausführung von entry-actions die höchste Priorität gegeben. Dadurch wird die Ausführung von entry-actions ohne Unterbrechung und dadurch die eindeutige Aktivierung des Zustands gewährleistet. Demzufolge wird während der Ausführung von entry-actions auf keine Austrittsbedingung reagiert.

Der zweite Teil dieser Regel beschreibt den Fall, in dem ein Zustandswechsel über eine Transition mit leerem Ereignis und leerer Bedingung erfolgen kann. Durch eine höhere Priorität für die Ausführungsbedingungen interner Aktionen findet der Zustandswechsel nicht statt, bevor diese Aktionen ausgeführt und abgeschlossen sind. Das Objekt geht in den Folgezustand über, nachdem die Aktivitäten des Zustands beendet sind. Ohne eine höhere Priorität für die Ausführungsbedingungen interner Aktionen würde die Austrittsbedingung mit leerem Ereignis und leerer Bedingung immer erfüllt sein und dadurch würden die internen Aktionen nie ausgeführt werden.

In dem dritten Teil wird die Unterbrechung von Aktionen des Zustands zugelassen, die nicht als entry-actions definiert worden sind. Während der Ausführung dieser Aktionen reagiert das Objekt auf die Ereignisse und die Bedingungen, die einen Zustandswechsel verursachen, indem das Objekt die Ausführung dieser Aktionen unterbricht und mit den Aktivitäten des Folgezustands anfängt. Dies entspricht dem Zustandswechsel gemäß in der UML [OMG03] definierter intuitiver Semantik für die Zustandsmaschinen.

Da die do-actions eines Zustands ausgeführt werden, solange das Objekt sich in diesem Zustand befindet, muss für Zustände mit do-actions eine Austrittsbedingung definiert werden, die durch den Austritt aus dem Zustand eine Unterbrechung der do-actions veranlassen, ansonsten werden die do-actions immer wieder wiederholt und der Zustand wird nicht verlassen. Daher wird in dem letzten Teil der Regel verlangt, dass sich die Austrittsbedingung eines Zustands mit do-actions oder mit Teilzuständen ohne einen Endzustand von dem leeren Ereignis und der leeren Bedingung unterscheiden muss.

5.6 Pseudozustände

In UML Zustandsdiagrammen werden einige Bezeichnungen verwendet, die Pseudozustände genannt werden. Diese Bezeichnungen werden nachfolgend mit ihrer Bedeutung definiert.

5.6.1 Startsymbol

Ein gefüllter Kreis, der mittels Defaulttransition mit dem Anfangszustand verbunden ist. Das Startsymbol wird verwendet, um unter nichtparallelen Zuständen den Anfangszustand zu kennzeichnen. Dieses Symbol ist kein Zustand und hat nicht die Semantik eines Zustands.

5.6.2 Endzustand

Ein umrandeter gefüllter Kreis, der bei nichtparallelen Zuständen das Ende der Aktivitäten des Objekts kennzeichnet. Daher kann ein Endzustand keine ausgehenden Transitionen haben.

In der vorliegenden Arbeit wird bei direkten Teilzuständen z_1 , ... , z_n und $z_1 \nparallel \ldots \nparallel z_n$ eines Zustands z der Endzustand mit *final*(z) bezeichnet. Bei Zuständen ohne einen Oberzustand wird ein Endzustand mit *final*() bezeichnet.

In einem Endzustand werden keine Aktionen und keine Teilzustände definiert. Formal heißt das:

$en(final(z)) = act(final(z)) = do(final(z)) = ex(final(z)) = a_\varepsilon$ und $include(final(z)) = \varnothing$.

In UML [OMG03] wird zwar für Endzustände verlangt, dass sie keine ausgehende Transition haben, das darf jedoch den Fall nicht mit ausschließen, dass die Austrittsbedingungen eines Zustands z auch für einen Endzustand gelten, der ein Teilzustand von z ist. Existiert für einen Zustand z ein Endzustand *final*(z), so gelten nach Abschnitt 5.3.6 alle Austrittsbedingungen des Zustands z auch für *final*(z), denn es gilt $final(z) \in include(z)$.

Wenn alle in z definierten Aktionenfolgen ausgeführt und abgeschlossen sind, befindet sich das Objekt in *final*(z), solange die Austrittsbedingung von z nicht erfüllt ist. In diesem Fall erfolgt das Verlassen von z aus dem Teilzustand *final*(z), sobald die Austrittsbedingung von z erfüllt ist. Daher gelten die ausgehenden Transitionen eines Zustands z auch für *final*(z).

5.6.3 History-State *H*(*z*)

Bei dem Verlassen eines aktiven Zustands mit Teilzuständen werden alle aktiven Teilzustände verlassen. Für einen Zustand z wird der zuletzt aktive direkte Teilzustand *History-State* genannt und mit $H(z)$ bezeichnet. In UML steht in der grafischen Notation ein umrandetes H für $H(z)$. Dabei ist $H(z) \in include(z)$ der Teilzustand von z, dessen exit-actions unmittelbar vor den exit-actions des Zustands z ausgeführt wurden. Für den Fall, dass z zuvor jedoch nicht aktiv war, kann ein bestimmter Teilzustand als History-State festgelegt werden. Dieser Zustand wird in der grafischen Notation durch einen Pfeil gekennzeichnet, der aus dem History-Symbol ausgeht. Beim Fehlen dieses Pfeils gilt $default(z)$ als History-State. Formal lässt sich der History-State $H(z)$, wie folgt, definieren:

5.6.3.1 Definition (History-State)

Es sei z ein Zustand mit direkten Teilzuständen z_1 , ... , z_n und $n \in \mathbb{N}$. Der History-State $H(z)$ ist, wie folgt, definiert:

1. Wenn $z_1 \nparallel ... \nparallel z_n$ gilt, dann gilt:

$$H(z) = \begin{cases} z_i \in include(z) & \text{falls} \quad en(z) \text{ und } ex(z_i); ex(z) \text{ wurden ausgeführt} \\ z_j \in include(z) & \text{falls} \quad en(z) \text{ wurde nicht ausgeführt und } z_j \text{ fest definiert} \\ default(z) & \text{sonst} \end{cases}$$

2. Wenn $z_1 \parallel ... \parallel z_n$ gilt, dann gilt $H(z) = \{H(z_1), ... , H(z_n)\}$.

Wenn $z_1 \nparallel ... \nparallel z_n$ gilt, dann wird im ersten Fall durch die Bedingung „$en(z)$ und $ex(z_i)$; $ex(z)$ wurden ausgeführt“ gewährleistet, dass z durch die Ausführung von $en(z)$ bereits aktiviert und durch die Ausführung von $ex(z_i)$; $ex(z)$ verlassen worden ist. In diesem Fall ist z_i der zuletzt aktive direkte Teilzustand von z, denn durch die unmittelbare Ausführung von $ex(z_i)$ vor $ex(z)$ wurde z_i vor z verlassen. In diesem Fall gilt $H(z) = z_i$. Die Bedingung „$en(z)$ wurde nicht ausgeführt“ bezeichnet den Fall, in dem z zuvor nicht aktiviert worden ist. In diesem Fall gilt $H(z) = z_j$, falls ein Zustand z_j für diesen Fall fest definiert ist. Schließlich gilt $H(z) = default(z)$, wenn z nicht bereits aktiviert wurde und kein Zustand z_j als History-State festgelegt worden ist.

Wenn $z_1 \parallel ... \parallel z_n$ gilt, dann liefert die Definition für $H(z)$ einen Zustand z', der aus parallelen Regionen besteht. Dabei ist jede parallele Region der History-State einer zu z gehörenden parallelen Region.

5.6.4 Deep-History-State $H^*(z)$

Die Teilzustände eines Zustands können selbst aus weiteren Teilzuständen bestehen. Bevor ein Zustand z verlassen wird, ist mindestens einer seiner Teilzustände aktiv. Dieser Teilzustand wird *Deep-History-State* genannt und mit $H^*(z)$ bezeichnet. In UML steht in der grafischen Notation ein umrandetes H^*, das Deep-History-Symbol genannt wird, für Deep-History-State.

Formal lässt sich die Definition des *Deep-History-State* eines Zustands z mit Hilfe von History-State rekursiv festlegen.

5.6.4.1 Definition (Deep-History-State)

Es sei z ein Zustand mit $in(z) \neq \varnothing$. Dann ist $H^*(z)$, wie folgt, definiert:

$$H^*(z) = \begin{cases} H(z) & \text{falls} \quad in(z) = include(z) \\ H^*(H(z)) & \text{sonst} \end{cases}$$

Bei dieser Definition ist der Deep-History-State für die Zustände, deren direkte Teilzustände keine weiteren Teilzustände besitzen, mit dem History-State identisch. In den anderen Fällen ergibt sich der Deep-History-State aus den Deep-History-States tieferer Teilzustände.

Die Zustände $H(z)$ und $H^*(z)$ sind im Fall von $in(z) = \varnothing$ undefiniert, das heißt, wenn ein Zustand z keine Teilzustände besitzt, dann hat er keinen History-State bzw. keinen Deep-History-State.

Nachdem die Begriffe Aktion, Aktionenfolge, Ereignis, Zustand und Zustandsübergang bzw. Transition definiert wurden, wird die Zustandsmaschine als das Fundament für die Zustandsdiagramme der UML definiert.

5.7 Definition (Zustandsmaschine)

Eine Zustandsmaschine $\mathcal{M}_O$ wird durch das Tupel $\mathcal{M}_O = (\mathcal{Z}, \mathcal{T}, default(), final())$ definiert. Dabei sind

1. $\mathcal{Z}$ eine endliche nichtleere Menge von Zuständen,
2. $\mathcal{T}$ eine endliche nichtleere Menge von Transitionen,
3. *default*() der Startzustand und
4. *final*() der Endzustand.

Mit Hilfe von Zustandsdiagrammen, die eine grafische Notation von Zustandsmaschinen sind, lassen sich die Lebenszyklen von einzelnen Komponenten in einem System

spezifizieren. Beispiele für die Zustandsdiagramme und für die Beschreibung des Verhaltens der Systemkomponenten sind in Kapitel drei gegeben.

Mit den in diesem Kapitel eingeführten Definitionen für Aktionen, Aktionenfolgen, Zustände und Transitionen lässt sich mit Hilfe von Zustandsdiagrammen der UML das Verhalten von Komponenten in einem Sicherungssystem präzise beschreiben. Um weitere in den Zustandsdiagrammen der UML existierende Konstrukte vorzustellen, werden im Abschluss dieses Abschnitts noch die weiteren Pseudozustände eines Zustandsdiagramms erwähnt. Weiterhin wird die Erweiterung der Zustandsdiagramme in der Version 2.0 der UML [OMG04] vorgestellt und die Äquivalenz dieser Erweiterung mit der hier eingeführten Definition gezeigt.

5.8 Weitere Pseudozustände

Um einige Vereinfachungen in die Notationen einzuführen, werden in den Zustandsdiagrammen der UML [OMG03 und OMG04] noch weitere Pseudozustände verwendet. Diese unterscheiden sich nicht von den hier eingeführten Zuständen. Daher werden nachfolgend diese Pseudozustände und ihre in der UML definierten Eigenschaften nur erwähnt, ohne formale Definitionen für sie zu geben.

5.8.1 Join

Join ist ein Pseudozustand mit mehreren eingehenden Transitionen und nur einer einzigen ausgehenden Transition. Er wird dazu verwendet, einen Wechsel von einem Zustand mit parallelen Regionen in einen anderen Zustand ohne parallelen Regionen zu modellieren. Dabei ist jede eingehende Transition eine ausgehende Transition eines Teilzustands in einer der parallelen Regionen.

5.8.2 Fork

Fork ist ein Pseudozustand mit einer einzigen eingehenden Transition und mehreren ausgehenden Transitionen. Fork wird verwendet, einen Übergang von einem einzelnen Zustand in einen Zustand mit parallelen Teilzuständen zu modellieren. Die mehrfachen ausgehenden Transitionen bestimmen die Zielzustände in den parallelen Regionen.

5.8.3 Junction und Choice

Für diese zwei Pseudozustände können eine oder mehrere eingehende und eine oder mehrere ausgehende Transitionen definiert werden. Dadurch ist es möglich, mehrere Transitionen zusammenzufügen bzw. unterschiedliche Folgezustände zu modellieren.

5.9 Zustandsdiagramme in UML 2.0

In UML 2.0 [OMG04] sind die Zustandsdiagramme um die Eigenschaft erweitert worden, dass für einen Zustand mehrfache entry-actions und mehrfache exit-actions definiert werden können. In der grafischen Notation werden jede entry-actions durch ein Entry-Symbol (einen leeren Kreis) bzw. jede exit-actions durch ein Exit-Symbol (einen Kreis mit einem Kreuz) gekennzeichnet, die am Rande des Zustands angezeigt werden. Jedes Symbol wird mit dem Namen der auszuführenden Aktion bzw. der Aktionenfolge beschriftet. Die alternative Schreibweise für mehrfache entry- bzw. exit-actions besteht darin, dass der Name der Aktion innerhalb zweier Halbkreise zusammen mit dem Transitionspfeil geschrieben wird. In diesem Fall werden die Entry- bzw. Exit-Symbole nicht mehr verwendet. In Abb. 5.12 hat der Zustand *z* drei entry-actions A_1 , A_2 , a_ε und drei exit-actions A_3 , A_4 , a_ε .

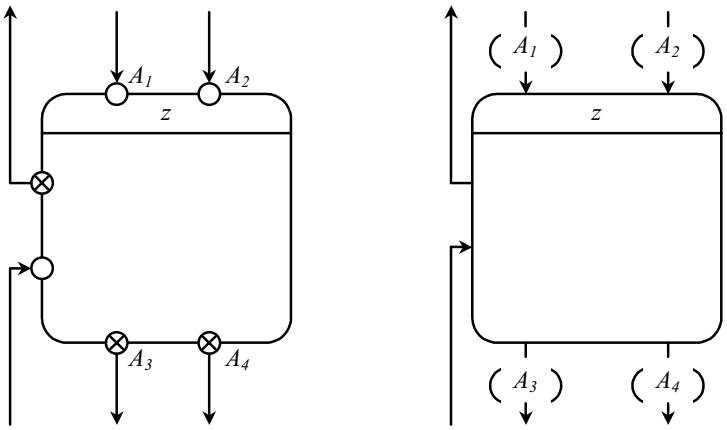


Abb. 5.12 Mehrfache entry- bzw. exit-actions für einen Zustand in UML 2.0

Durch diese Erweiterung kann für einen Zustand, der über verschiedene Wege erreicht bzw. verlassen wird, die alternative Ausführung von entry- bzw. exit-actions eingeführt werden. Folglich können in dem Zustand abhängig von jeder Transition, die den Eintritt in den Zustand oder den Austritt aus dem Zustand verursacht, entsprechende entry- oder exit-actions definiert werden.

Hier wird gezeigt, dass die genannte Möglichkeit auch durch die hier eingeführte Definition gegeben ist.

5.9.1 Äquivalenz von Zustandsdiagrammen der UML 1.5 und 2.0

Es sei *z* ein Zustand. Weiter seien e_1 [b_1] / A_1 , ... , e_n [b_n] / A_n mit $n > 1$ Beschriftungen der Transitionen, , die die Eintritte in *z* ermöglichen. Bei dem Eintritt über die Tran-

sition mit der Beschriftung e_i [b_i] / A_i und $1 \leq i \leq n$ seien en_i (z) = A'_i die entry-actions gemäß UML 2.0.

Auch ohne die Anwendung mehrfacher entry-actions lässt sich die in Abhängigkeit von einzelnen Transitionen definierte Ausführung von A'_i durch das Einführen einzelner Zustände für jede A'_i modellieren.

Entsprechend jeder en_i (z) mit $1 \leq i \leq n$ wird ein Zustand z'_i mit $en(z'_i) = ex(z'_i) = a_\varepsilon$ definiert, in dem lediglich A'_i ausgeführt wird und alle anderen Aktivitäten aus leeren Aktionen bestehen. Für z'_i gilt dann:

1. z'_i und z haben den gleichen direkten Oberzustand,
2. Die Transition mit der Beschriftung e_i [b_i] / A_i führt in z'_i und
3. Eine Transition mit der Beschriftung e_ε [b_ε] / a_ε führt von z'_i in z.

Da die Transition mit der Beschriftung e_i [b_i] / A_i in z'_i führt, wird gemäß Abschnitt 5.4.2 A_i ausgeführt. Unmittelbar danach wird durch den Eintritt in z'_i die Aktionenfolge A'_i ausgeführt. Diese Aktionenfolge wird ohne eine Unterbrechung zu Ende geführt, weil gemäß der unter 5.5.3 genannten Regel der Austritt aus z'_i durch die Transition mit der Beschriftung e_ε [b_ε] / a_ε veranlasst wird. Danach erfolgt der Eintritt in z und es wird mit den weiteren Aktivitäten des Zustands z fortgesetzt.

Sind die Transitionen mit Beschriftungen e_1 [b_1] / A_1 , ... , e_n [b_n] / A_n und $n > 1$ die Transitionen, die den Austritt aus z veranlassen und sind ex_i (z) = A''_i die exit-actions, die abhängig von der Transition mit der Beschriftung e_i [b_i] / A_i und $1 \leq i \leq n$ ausgeführt werden, so wird ein Zustand z''_i mit $en(z''_i) = ex(z''_i) = a_\varepsilon$ definiert, in dem lediglich A_i ausgeführt wird und alle anderen Aktivitäten aus leeren Aktionen bestehen. Für z''_i gilt dann:

1. z''_i und z haben den gleichen direkten Oberzustand,
2. Die Transition mit der Beschriftung e_i [b_i] / A''_i führt von z in z''_i und
3. Die Transition mit der Beschriftung e_ε [b_ε] / a_ε veranlasst den Austritt aus z''_i .

Da die Transition mit der Beschriftung e_i [b_i] / A''_i von z in z''_i führt, wird gemäß Abschnitt 5.4.2 unmittelbar vor dem Eintritt in z''_i die Aktionenfolge A''_i ausgeführt. Danach wird durch den Eintritt in z''_i die Aktionenfolge A_i ausgeführt. Die Ausführung von A_i erfolgt ohne eine Unterbrechung, weil gemäß der unter 5.5.3 genannten Regel der Austritt aus z''_i durch die Transition mit der Beschriftung e_ε [b_ε] / a_ε veranlasst wird.

Außer Zustandsdiagrammen gibt es in UML2.0 [OMG04] noch die Protokollautomaten. Diese Automaten werden insbesondere für die Beschreibung einer Sequenz von Transitionen verwendet, die z. B. durch ein Objekt veranlasst werden können. In der vorliegenden Arbeit wird jedoch auf diese Automaten nicht eingegangen.

5.10 Prozesse und Interaktionen

Mit Hilfe der bis jetzt diskutierten Zustandsdiagramme der UML kann das interne Verhalten von Komponenten in Form der Ausführung von Aktivitäten beschrieben werden. Der Schwerpunkt dieser Beschreibung liegt darin darzustellen, welche Aktionen die einzelnen Komponenten ausführen, auf welche Ereignisse sie reagieren und welche Zustände sie in ihrem Lebenszyklus einnehmen. Für die Spezifikation des dynamischen Verhaltens des Systems ist auch die Beschreibung der Prozesse und Interaktionen zwischen den Systemkomponenten notwendig.

Für die Beschreibung der Interaktionen zwischen den Systemkomponenten stellt die UML [OMG03] die Sequenzdiagramme zur Verfügung. Wie in Kapitel drei gezeigt wurde, können diese Diagrammarten dafür verwendet werden, den Ablauf der Prozesse zu spezifizieren.

In einem Sequenzdiagramm werden für die an einem Prozess beteiligten Objekte die sogenannten Lebenslinien eingeführt, die eine Zeitachse darstellen. Zwischen den Lebenslinien werden die auszutauschenden Informationen in Form von Pfeilen dargestellt, die vom Sender zum Empfänger führen.

Trotz der Übersichtlichkeit der Beschreibung von Interaktionen zwischen den Objekten gibt es bei der Anwendung von Sequenzdiagrammen einige Einschränkungen. Einige dieser Einschränkungen liegen an einer fehlenden Möglichkeit zur Darstellung von

1. Informationen, die von einem Objekt an mehrere Objekte gesendet werden,
2. Vorbedingungen für die Abwicklung eines Prozesses,
3. der Wiederholung der Vorgänge,
4. optionalem Verhalten für einen Prozess,
5. alternativem Verhalten für einen Prozess,
6. bereits spezifiziertem Vorgang innerhalb eines Prozesses,
7. paralleler Ausführung von Interaktionen innerhalb eines Prozesses.

Für die Darstellung einer Nachricht, die von einem Objekt an mehrere Objekte gesendet wird, wurde in UML-RT (UML Real Time), wie z. B. in [BK+02], ein spezieller Pfeil eingeführt. Dieser Pfeil hat an der Seite des Senders einen kleinen Kreis und schneidet die Lebenslinien aller Empfänger.

Für die Beschreibung der Vor- und Nachbedingungen eines Prozesses werden in Live Sequence Charts (LSC) [DH01, BD+04], die eine formal fundierte Form von Sequenzdiagrammen sind, weitere LSCs verwendet. In LSCs gibt es weiterhin die Möglichkeit zu bestimmen, ob ein Sachverhalt auftreten muss oder ob er nur auftreten kann.

Für die weiteren genannten Punkte sind in UML 2.0 [OMG04] Erweiterungen in Form von geeigneten Operatoren eingeführt, die Spezifikation einer optionalen, alternativen,

wiederholten oder parallelen Ausführung von Interaktionen ermöglichen. Weiter gibt es in UML 2.0 die Möglichkeit, die Sequenzdiagramme ineinander zu referenzieren oder die Reihenfolge der Ausführung verschiedener Interaktionen festzulegen.

In dem folgenden Abschnitt wird für die Sequenzdiagramme der UML eine formale Definition präsentiert. Auf Basis dieser Definition werden die in der Systemanforderungsspezifikation spezifizierten Prozesse präzisiert. Weiter bildet die folgende Definition eine Basis für die Integration der Systemprozesse in die anschließende formale Verifikation des Systems, die in Kapitel sieben ausgeführt wird.

5.11 Sequenzdiagramme

In einem Sequenzdiagramm werden die Interaktionen unter Systemkomponenten in Form von Szenarios definiert. Ein Szenario ist der Informationsaustausch unter Objekten in einem bestimmten Zeitraum ihres Lebenszyklus. Daher entsprechen die Angaben in einem Sequenzdiagramm einem temporären Verhalten der beteiligten Objekte.

In einer formalen Definition eines Sequenzdiagramms wird die auszutauschende Information in Form einer Aktion a dargestellt, die von einem Objekt O_i und eventuell unter einer bestimmten Bedingung ausgeführt wird. Von dem Ergebnis dieser Aktion erfährt das Objekt O_j .

Um die temporäre Ordnung unter den auszutauschenden Informationen darstellen zu können, wird der Begriff Position eingeführt, die als eine natürliche Zahl die Position einer Interaktion in dem Szenario bestimmt.

Damit lässt sich ein Sequenzdiagramm, wie folgt, definieren:

5.12 Definition (Szenario)

Ein Szenario ist ein Tupel ($\mathcal{N}$, $\mathcal{B}_{SD}$, P) mit folgenden Eigenschaften:

1. $\mathcal{N}$ ist eine endliche nicht leere Menge von Interaktionen (O_i .[b] a, O_j , p), wobei O_i , $O_j \in M$, $b \in \mathcal{B}_{SD}$ und $a \in \mathcal{A}_{O_i}$ sind und bei $i \neq j$ zusätzlich $a \in \mathcal{E}_{O_j}$ gilt. Weiter ist $p \in P$ eine Position.
2. $\mathcal{B}_{SD} \subseteq \mathcal{B}_{O_i} \cup \mathcal{B}_{O_j}$ ist eine endliche Menge von Bedingungen.
3. $P \subseteq \mathbb{N}$ ist eine endliche nicht leere Menge von Positionen.

Eine Interaktion (O_i .[b] a , O_j , p) beschreibt die Ausführung der Aktion a unter der Bedingung b von dem Objekt O_i , wobei von dem Ergebnis dieser Aktion das Objekt O_j

erfährt. Die natürliche Zahl p zeigt, die Stelle in dem Szenario, an der die Interaktion stattfindet.

5.13 Definition (Ordnungsrelation zwischen Interaktionen)

Bei zwei Interaktionen $(O_i.[b]\ a, O_j, p)$ und $(O_k.[b']\ a', O_r, p')$ gilt $p \leq p'$ genau dann, wenn die Aktionen a und a' entweder gleichzeitig oder zuerst a und danach a' ausgeführt werden. Damit gilt in $\mathcal{N}$ die Ordnungsrelation „ $\preccurlyeq$ “ mit

$$(O_i.[b]\ a, O_j, p) \preccurlyeq (O_k.[b']\ a', O_r, p') \Leftrightarrow p \leq p'.$$

Mit Hilfe der Zustands- und Sequenzdiagramme der UML lässt sich das dynamische Verhalten eines Systems beschreiben. Für die Beschreibung des Systemaufbaus und der Beziehungen zwischen den Systemkomponenten werden die Klassendiagramme der UML und die darin enthaltenen Relationen verwendet. Im folgenden Kapitel werden diese Relationen diskutiert und formal fundiert.

6 Systemstruktur

In einer Systemspezifikation wird die Systemstruktur anhand der Darstellung der Systemkomponenten und ihrer Beziehungen untereinander beschrieben. Hierfür werden in der objektorientierten Analyse die Relationen verwendet, die unter den Objekten und mit der Systemumgebung definiert werden. Diese Beziehungen spielen sowohl in der Struktur als auch in dem Verhalten der Objekte eine wichtige Rolle. Eine präzise Beschreibung dieser Beziehungen setzt voraus, dass die Relationen, die diese Beziehungen darstellen, eindeutig definiert und interpretiert werden.

6.1 Relationen

Eine Relation beschreibt die Beziehung zwischen zwei oder mehreren Objekten eines Objektsystems. UML [OMG03] stellt die Relationen Assoziation, Komposition, Aggregation und Vererbung zur Verfügung. Mit Hilfe dieser Relationen können für jedes Objekt seine Kommunikationspartner, seine Teilkomponenten und die Objekte identifiziert werden, die gemeinsame Eigenschaften mit dem Objekt teilen. Diese Relationen werden in einem Klassendiagramm festgelegt und werden dafür eingesetzt, die Struktur und den Aufbau des Systems sowie die Implikationen innerhalb des Systems zu spezifizieren.

In einem Klassendiagramm wird eine Relation im Kontext der Klassen definiert und drückt das Verhältnis zwischen den Objekten der beteiligten Klassen aus.

Hier wird der Begriff Relation präzisiert, indem eine formale Schreibweise und eine Definition eingeführt werden.

6.1.1 Bezeichnung

In der textuellen Notation wird hier eine Relation *rel* zwischen zwei Klassen K^p und K^q in Form von (nK^p *rel* mK^q) notiert und es gelten (nK^p *rel* mK^q) $\in \mathcal{R}_{K^p}$ und (nK^p *rel* mK^q) $\in \mathcal{R}_{K^q}$. Dabei sind *n* und *m* natürliche Zahlen und zeigen, dass *n* Objekte der Klasse K^p mit *m* Objekten der Klasse K^q in Beziehung *rel* stehen.

Formal hat dieser Ausdruck die folgende Bedeutung:

6.1.2 Definition (Relation)

Je *n* Objekte der Klasse K^p stehen mit je *m* Objekten der Klasse K^q in Relation *rel* (nK^p *rel* mK^q), wenn es für eine Teilmenge $\{K^p_{O_i}, \ldots, K^p_{O_n}\}$ von Objekten der Klasse K^p eine Teilmenge $\{K^q_{O_l}, \ldots, K^q_{O_m}\}$ von Objekten der Klasse K^q gibt, so dass ($K^p_{O_i}$ *rel* $K^q_{O_j}$) für $0 \le i \le n$ und $0 \le j \le m$ gilt.

n und *m* können auch unbestimmte natürliche Zahlen sein. UML [OMG03] verwendet das Symbol * für eine unbestimmte natürliche Zahl inklusive Null. Ebenso sind für *n* und *m* Notationen der Form *r..k* bzw. *r..** möglich, die für die Angabe einer natürlichen Zahl *i* mit $r \le i \le k$ bzw. $r \le i$ angewendet werden.

Sind $n = 1$ und $m = 1$, so wird auf die Zahlenangabe verzichtet und einfach die Schreibweise (K^p *rel* K^q) verwendet. Bei einem Element O_i *rel* O_j in den Mengen der Relationen von O_i und O_j wird daher keine Anzahl angegeben, denn dabei handelt es sich jeweils um ein Objekt.

Mit dieser Beschreibung lassen sich nun die Menge der Relationen einer Klasse und die Menge der Relationen eines Objekts aus der Definition 4.4 präzisieren.

6.1.3 Definition (die Menge der Relationen von Klassen bzw. Objekten)

Die Menge $\mathcal{R}_{K^p}$ der Relationen einer Klasse K^p und die Menge $\mathcal{R}_O$ der Relationen eines Objekts *O* werden, wie folgt, definiert:

$\mathcal{R}_{K^p} = \{(nK^p \; rel \; mK^q)$ oder $(nK^q \; rel \; mK^p) \mid rel$ ist eine Relation und $n, m \in \mathbb{N}\}$ bzw.

$\mathcal{R}_{\mathcal{O}} = \{(O \; rel \; O_i)$ oder $(O_i \; rel \; O) \mid rel$ ist eine Relation$\}$.

Im Allgemeinen folgt aus einer Relation zwischen einer Klasse und einer anderen Klasse nicht, dass jedes Objekt der zweiten Klasse einem entsprechenden Objekt der ersten Klasse zugeordnet werden kann, so dass die Relation zwischen dem Objekt der ersten Klasse und dem Objekt der zweiten Klasse gilt.

Ein Beispiel dafür ist das Verhältnis zwischen den Objekten der Klasse *Fahrzeug* und den Objekten der Klasse *Kommunikationsmodul*. Jedes Objekt der Klasse *Fahrzeug* hat als Teil ein Objekt der Klasse *Kommunikationsmodul*, jedoch nicht jedes Objekt der Klasse *Kommunikationsmodul* ist ein Teil von einem Objekt der Klasse *Fahrzeug*.

In den nächsten Abschnitten werden Relationen aus dem objektorientierten Ansatz basierend auf den Eigenschaften der Beziehungen zwischen den Systemkomponenten, wie sie in einer Systemdefinition vorkommen, und ausgehend von ihrer Beschreibung in UML [OMG03] charakterisiert.

6.2 Kommunikationsbeziehung

Ein Objektsystem ist eine Sammlung von kommunizierenden Objekten. Zur Spezifikation eines Objektsystems gehören unter anderem die Identifizierung und die Beschreibung der Kommunikationen unter den Objekten. In einem objektorientierten Modell können Assoziationen verwendet werden, um diese Beziehung zu beschreiben.

In UML [OMG03] wird eine Assoziation als die Beschreibung einer Verbindung von einem Objekt zu einem oder mehreren anderen definiert. [Boo97] charakterisiert diese Verbindung mit der Angabe einer semantischen Abhängigkeit ohne Bezeichnung einer Richtung für diese Abhängigkeit und ohne Angabe einer exakten Art und Weise, wie ein Objekt mit einem anderen in Beziehung steht. In UML [OMG03] sind Assoziationen auch zwischen den Objekten derselben Klasse zulässig [Bal00].

Ein verteiltes System wie FFB besteht aus Komponenten, die untereinander Informationen austauschen und voneinander Dienste verlangen, ohne miteinander integriert oder direkt verbunden zu sein. Fahrzeuge, Fahrwegelemente und die FFB-Zentrale sind Beispiele für solche Komponenten. Um zu kennzeichnen, dass diese Objekte miteinander kommunizieren und voneinander Dienste verlangen, werden Assoziationen verwendet.

Der Austausch von Informationen und die Abwicklung von Diensten verlangen eine Zusammenarbeit der beteiligten Objekte. In einer objektorientierten Umgebung findet diese Zusammenarbeit ereignisgesteuert statt. Das bedeutet, dass ein Objekt eine Aktion

ausführt, die für ein anderes Objekt als ein Ereignis gilt. Danach reagiert das zweite Objekt auf dieses Ereignis entsprechend seines definierten Verhaltens.

Aus dieser Betrachtung folgt, dass es unter den Objekten, die an einer Assoziation beteiligt sind, Gemeinsamkeiten zwischen der Menge der Aktionen eines Objekts und der Menge der Ereignisse des anderen Objekts gibt. Diese Eigenschaft lässt sich im Kontext der Klassen, wie folgt, charakterisieren:

6.2.1 Definition (Assoziation)

> Es seien $K_{O_i}^{p}$ und $K_{O_i}^{q}$ zwei Objekte mit $\mathcal{A}_{K^p}$ Menge der Aktionen von K^p und $\mathcal{E}_{K^q}$ Menge der Ereignisse von K^q. Dann ist $K_{O_i}^{p}$ mit $K_{O_i}^{q}$ assoziiert ($K_{O_i}^{p}\ asso\ K_{O_i}^{q}$), wenn $\mathcal{A}_{K^p} \cap \mathcal{E}_{K^q} \neq \emptyset$ gilt.

Diese Beschreibung sagt, dass für eine Assoziation unter Objekten zweier Klassen Gemeinsamkeiten zwischen der Menge der Aktionen von der einen und der Menge der Ereignisse von der anderen bestehen müssen. Da für ein Objekt die Ereignisse Aktionen sind, die die anderen Objekte ausführen, sind diese Gemeinsamkeiten genau die Aktionen, die von Objekten einer Klasse der Assoziation ausgeführt werden und für die Objekte der anderen Klasse als Ereignis gelten.

Im Modell des FFB gibt es eine derartige Gemeinsamkeit z. B. zwischen den Klassen *Fahrzeug* und *Fahrwegelement*. Die Aktion *status_abfragen*() gehört der Menge der Aktionen der Klasse *Fahrzeug* $\mathcal{A}_{Fahrzeug}$ und der Menge der Ereignisse der Klasse *Fahrwegelement* $\mathcal{E}_{Fahrwegelement}$ an. Aufgrund dieser Gemeinsamkeit existiert in dem Klassendiagramm in Abb. 6.1 eine Assoziation mit dem Namen *kommuniziert* unter den Klassen *Fahrzeug* und *Fahrwegelement*.

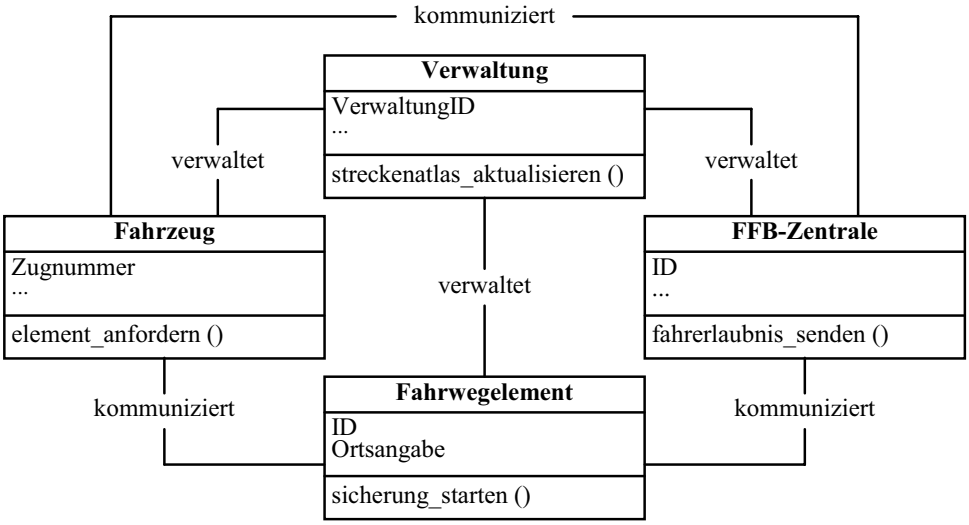


Abb. 6.1 Assoziationen in einem Klassendiagramm

Im Allgemeinen kann für zwei Klassen K^p und K^q eine Assoziation (nK^p *asso* mK^q) angegeben werden. Diese bedeutet, dass *n* Objekte der Klasse K^p mit *m* Objekten der Klasse K^q assoziieren.

6.2.2 Bidirektionale Assoziation

Es seien $K^p_{O_i}$ *asso* $K^q_{O_i}$ und $\mathcal{E}_{K^p}$ Menge der Ereignisse von K^p und $\mathcal{A}_{K^q}$ Menge der Aktionen von K^q. Wenn $\mathcal{E}_{K^p} \cap \mathcal{A}_{K^q} \neq \emptyset$ gilt, dann ist die Assoziation *asso* bidirektional.

6.2.2.1 Folgerung:

Da nicht jede Assoziation unbedingt bidirektional ist, lässt sich sofort schließen, dass eine Assoziation keine symmetrische Relation ist. Ebenso lässt sich die Transitivität ausschließen, denn aus den Gemeinsamkeiten der Mengen der Ereignisse und Aktionen von den Klassen K^p und K^q sowie den Klassen K^q und K^r folgern nicht unbedingt Gemeinsamkeiten der Mengen der Ereignisse und der Aktionen der Klassen K^p und K^r.

In UML [OMG03] sind Assoziationen auch zwischen den Objekten derselben Klasse zulässig. Im FFB sind die Kommunikationsmodule, über die die Betriebskomponenten miteinander kommunizieren, Beispiele für Objekte einer Klasse, die miteinander assoziiert sind. Daraus folgt:

6.2.3 Sonderfall

Gilt $\mathcal{A}_{K^p} \cap \mathcal{E}_{K^p} \neq \emptyset$, so assoziieren die Objekte der Klasse K^p miteinander.

Da es sich hier um einen Sonderfall handelt, gilt die folgende Folgerung.

6.2.3.1 Folgerung:

Eine Assoziation ist keine reflexive Relation.

Eine Assoziation ist auch zwischen mehreren Objekten möglich und kann, wie folgt, charakterisiert werden.

6.2.4 Verallgemeinerung:

Es seien K^1, ... , K^n Klassen mit $\mathcal{A}_{K^1}$, ... , $\mathcal{A}_{K^n}$ bzw. $\mathcal{E}_{K^1}$, ... , $\mathcal{E}_{K^n}$ Mengen der Aktionen bzw. Mengen der Ereignisse, auf die die Objekte dieser Klassen reagieren. Eine Assoziation ist unter K^1 bis K^n genau dann gegeben, wenn gilt:

$\exists\ \mathcal{A}_{K^i}$ mit $\mathcal{A}_{K^i} \cap (\bigcap_{j=1}^{n} \mathcal{E}_{K^j}) \neq \emptyset$, für alle $1 \leq i, j \leq n$.

Diese Beschreibung sagt, dass für eine Assoziation unter mehreren Klassen eine Klasse existieren muss, deren Objekte eine Aktion ausführen, auf die die Objekte anderer Klassen reagieren.

6.2.5 Folgerung

Wie in [Ste01] erwähnt wird, geht auch aus der Definition 6.2.1 hervor, dass bei der Anwendung von Assoziationen Fälle vorkommen können, in denen Objekte indirekt ineinander Aktivitäten auslösen, die bei der Modellierung nicht geplant sind. Das kann für drei Objekte O_1 , O_2 und O_3 mit (O_1 *asso* O_2), (O_2 *asso* O_3) und ohne (O_1 *asso* O_3) passieren, wenn eine Aktion in O_1 , die ein Ereignis für O_2 ist, in O_2 eine Aktion auslöst, die ihrerseits ein Ereignis für O_3 ist und in O_3 zur Ausführung einer Aktivität führt. Dadurch wird durch eine Aktion von O_1 eine Aktivität in O_3 ausgelöst, während in der Modellierung keine Assoziation zwischen O_1 und O_3 definiert wurde. Daher sollte bei der Modellierung die Auswirkung der Aktionen eines Objekts auf das Verhalten weiterer Objekte untersucht werden, wenn zwischen mehreren Objekten paarweise Assoziationen definiert worden sind.

6.3 Zusammengehörigkeit unter den Systemkomponenten

Bei der Darstellung der Systemstruktur ist eine der weiteren Anwendungen der Relationen die Beschreibung der Beziehung unter den Systemkomponenten und ihren Teilkomponenten.

Besteht unter Objekten zweier oder mehrerer Klassen eine physikalische oder konzeptuelle Zusammengehörigkeit, so wird diese Relation mit Aggregation oder Komposition dargestellt. In UML [OMG03] werden diese Relationen in den Klassendiagrammen durch eine Verbindungslinie bezeichnet, die an dem Ende zum Aggregat eine Raute besitzt. Ist die Raute leer bzw. gefüllt, so handelt es sich um eine Aggregation bzw. eine Komposition.

[Bal00] und [Oes99] unterscheiden diese Begriffe dadurch, dass bei einer Aggregation die Teilobjekte gleichzeitig mehreren Ganzen zugeteilt werden dürfen. Dabei können die Ganzen Instanzen unterschiedlicher Klassen oder Instanzen derselben Klasse sein. Bei einer Komposition dürfen die Teilobjekte zu jedem Zeitpunk ausschließlich zu einem einzigen Ganzen gehören. Zu einem anderen Zeitpunkt können sie aber einem anderen Ganzen zur Verfügung stehen. Weiter gilt für eine Komposition, dass beim Kopieren bzw. Löschen des Ganzen alle Teilobjekte mitkopiert bzw. mitgelöscht werden.

Weiter werden in UML [OMG03] Aggregation und Komposition transitive Relationen genannt.

In dieser Interpretation ist jedoch offen, wie die Teilobjekte bei einer Aggregation erzeugt oder gelöscht werden und wie die Kommunikation unter dem Ganzen und den Teilobjekten organisiert ist. Ferner ist ungeklärt, wie in einer Komposition die Teilobjekte eines bereits gelöschten Ganzen weiterbestehen können, um zu späteren Zeitpunkten für die anderen Ganzen verfügbar zu sein. Ebenso ist ungeklärt, wie eine Transitivität bei diesen Relationen herzuleiten ist.

Bei vielen Werken über formale objektorientierte Spezifikationsmethoden, wie [Ehr99, DeHa97, Rum96], wird nur eine Relation zur Beschreibung der Beziehung zwischen dem Ganzen und seinen Teilen verwendet, die entweder Komposition oder Aggregation genannt wird. Dabei wird nicht auf die Art der Zusammengehörigkeit der Systemkomponenten eingegangen.

Einige Autoren setzen als essentielle Eigenschaft dieser Relationen voraus, dass das Ganze das Verhalten der Teilobjekte steuert [EF+98, Oes99]. In diesem Fall ist eine Kommunikation mit den Teilobjekten nur über eine Kommunikation mit dem Ganzen möglich. Dabei verfügt dann das Ganze über Aktionen, die entsprechende Aktivitäten bei den Teilobjekten aufrufen, welche die gewünschte Funktionalität erbringen. Dieser Aspekt ist eine essentielle und entscheidende Eigenschaft dieser Relationen bei Softwaresystemen und kann in allen Entwicklungsphasen von der Analyse bis hin zur Implementierung verlangt werden. Er ist jedoch bei der Analyse von real existierenden Komponenten eines Systems, die nicht nur aus Software, sondern aus mehreren technischen Anlagen bestehen, ungeeignet. Unter dieser Annahme lassen sich dann die Komponenten nicht beschreiben, die zu einem Ganzen gehören und als Kommunikationsweg zu anderen Systemkomponenten gelten. Beispiele für diesen Fall sind das Kommunikationsmodul oder die Ausschaltsensorik eines Bahnübergangs. Diese Komponenten werden nicht nur über den Bahnübergang, sondern auch von den Fahrzeugen oder der FFB-Zentrale direkt angesprochen und ermöglichen für den Bahnübergang eine Kommunikation mit anderen Systemkomponenten.

In [BH99] werden die wichtigsten Charakteristika der Beziehungen unter einem Ganzen und seinen Teilen auf Basis ihrer ontologischen Aspekte erläutert. [Paz00] betrachtet diese Relationen in Anwendung auf die Zustandsdiagramme, die das dynamische Verhalten der beteiligten Objekte beschreiben.

In der vorliegenden Arbeit basiert die Beschreibung der Relationen Aggregation und Komposition auf den Beziehungen zwischen den Systemkomponenten im FFB. Dabei werden auf Basis dieser Verhältnisse Definitionen eingeführt, die in einer Systemdefinition für die Beschreibung der Beziehungen zwischen den Systemkomponenten und ihren Teilkomponenten verwendet werden können.

Im FFB existieren unterschiedliche Arten der Zusammengehörigkeit von Objekten zueinander. In einigen Fällen liegt eine strenge Form von Zusammengehörigkeit vor, in-

dem die Objekte und ihre Teilobjekte existenz- und verhaltensabhängig sind und die Teilobjekte in ihrem ganzen Lebenszyklus lediglich zu einem einzigen Objekt gehören.

Es existieren andere Fälle, in denen die Teilobjekte zur gleichen Zeit oder zu unterschiedlichen Zeiten zu weiteren Objekten gehören können.

Die folgenden Beispiele zeigen die Charakteristika dieser Beziehungen.

6.3.1 Beispiel (Fahrweg)

Im FFB besteht ein Fahrweg aus zustandsvariablen Fahrwegelementen, Gleisabschnitten und Balisen. Wird einem Fahrzeug ein Fahrweg zugewiesen, so werden ihm auch die dazugehörigen Fahrwegelemente und Gleisabschnitte zugewiesen. Der Fahrweg wird ganz oder teilweise freigegeben, wenn die genannten Teilobjekte von dem Fahrzeug ganz oder teilweise freigegeben sind. Nach der Freigabe der einzelnen Teilobjekte bzw. nach einer eventuellen Rückgabe des zugewiesenen Fahrwegs können die Teilobjekte für die Bildung weiterer möglicher Fahrwege eingesetzt werden. Bei einem Fahrweg kommunizieren die Teilobjekte weder unter sich noch mit dem Fahrweg selbst. Auch die Statusänderungen des Fahrwegs, die sich aus den Zuständen der jeweiligen Teilobjekte ergeben, erfolgen nicht über einen direkten Informationsaustausch zwischen dem Fahrweg und seinen Teilkomponenten, sondern indirekt über eine Kommunikation, die zwischen dem Fahrzeug und der FFB-Zentrale stattfindet. Weiter gilt, dass die Komponenten eines Fahrwegs ohne ihn weiter bestehen können, denn nach Auflösung eines Fahrwegs bleiben seine Komponenten im System erhalten, und sie werden für die Bildung anderer Fahrwege benutzt.

Dieses Beispiel zeigt, dass es unter den Objekten auch Zusammengehörigkeiten geben kann, in denen keine Abhängigkeit von Existenz oder Verhalten zwischen dem Ganzen und seinen Teilen besteht.

In solchen Fällen bleibt die Verfeinerung der Beziehung zwischen dem Ganzen und seinen Teilen bis in die Entwurfsphase offen. In der Entwurfsphase kann die Beziehung unter den beteiligten Objekten entsprechend den Realisierungsentscheidungen verfeinert werden.

In dem folgenden Beispiel wird eine andere Art von Zusammengehörigkeit gezeigt, in der die Beziehung unter dem Ganzen und seinen Teilen andere Charakteristika hat.

6.3.2 Beispiel (Bahnübergang)

Fast bei allen technischen Anlagen (Bahnübergängen, Weichen etc.) besteht eine existenzabhängige und exklusive Zusammengehörigkeit unter dem Ganzen und seinen Teilen. So gilt für einen Bahnübergang, der im objektorientierten Modell eine Schrankenanlage, eine Lichtzeichenanlage, eine Ausschaltsensorik, ein Kommunikationsmodul

und einen Timer besitzt, dass die Teilobjekte nicht ohne das Ganze im System existieren können. Diese Teilkomponenten existieren für ein Verkehrssystem stets im Zusammenhang mit einem Bahnübergang. Wird ein Bahnübergang in bzw. außer Betrieb genommen, so stehen dann alle seine Teilobjekte für alle bzw. keine weiteren Systemkomponenten zur Verfügung. Im Kontext der Systembeschreibung gehören die Teilkomponenten in ihrer ganzen Lebensdauer zu einem einzigen Bahnübergang. Das bedeutet, dass eine strenge existenzabhängige und exklusive Beziehung unter dem Ganzen und seinen Teilen herrscht.

Ferner werden die Teilkomponenten lediglich von dem Bahnübergang gesteuert. Die Statusänderungen des Bahnübergangs und seiner Teilobjekte hängen gegenseitig voneinander ab und erfolgen über einen direkten Informationsaustausch. Sogar die Teilobjekte, wie die Ausschaltsensorik und das Kommunikationsmodul, die auch mit anderen Objekten im Gesamtsystem wie mit Fahrzeugen oder mit der FFB-Zentrale kommunizieren, müssen zuerst von dem Bahnübergang initiiert bzw. zur Verfügung gestellt werden. Das zeigt eine strenge verhaltenabhängige Beziehung unter dem Ganzen und seinen Komponenten.

Diese Beispiele enthalten nicht alle Charakteristika der verschiedenen Arten von Zusammengehörigkeiten, die unter den Komponenten eines Systems möglich sind. Anhand dieser Beispiele soll der Fall mit einer exklusiven und existenzabhängigen Zusammengehörigkeit der Komponenten eines Systems demonstriert werden, der in der Systemdefinition relevant ist.

Ausgehend von diesen Beispielen werden die Relationen Aggregation und Komposition charakterisiert.

6.3.3 Aggregation

Für eine physikalische oder konzeptuelle Zusammengehörigkeit unter zwei oder mehreren Objekten wird die Relation Aggregation verwendet. Diese Relation zeigt lediglich, dass es unter den beteiligten Objekten eine Zusammengehörigkeit gibt. Sie beinhaltet keine weiteren Eigenschaften, die auf eine Abhängigkeit von Existenz oder Verhalten unter den Objekten hindeuten. Ebenso trifft diese Relation keine Aussage darüber, ob die Teile nur zu einem oder zu mehreren Ganzen gehören.

6.3.4 Definition (Aggregation)

> Je n Objekte der Klasse K^p sind Aggregate von je m Objekten der Klasse K^q ($nK^p \ aggr \ mK^q$), wenn es für eine Teilmenge $\{K^p_{O_i}, \ldots, K^p_{O_n}\}$ von Objekten der Klasse K^p eine Teilmenge $\{K^q_{O_i}, \ldots, K^q_{O_m}\}$ von Objekten der Klasse K^q gibt, so dass ($K^p_{O_i} \ aggr \ K^q_{O_j}$) für $0 \le i \le n$ und $0 \le j \le m$ gilt.

Wenn $K^q_{O_1}$, ... , $K^q_{O_m}$ Teile von $K^p_{O_1}$ sind, dann sind $K^p_{O_1}$ *aggr* $K^q_{O_1}$, ... , $K^p_{O_1}$ *aggr* $K^q_{O_m}$ Elemente der Menge der Relationen von $K^p_{O_1}$ und $K^q_{O_1}$, ... , $K^q_{O_m}$.

Durch diese Definition ist es möglich, die Zusammengehörigkeit der Objekte zu bezeichnen, ohne auf die weiteren Eigenschaften dieser Zusammengehörigkeit einzugehen. Bei der Anwendung dieser Relation in der Systemspezifikation bleibt eine genaue Spezifikation der Art dieser Beziehung bis in die Designphase offen. Daher kann diese Relation bei den Fällen verwendet werden, in denen weitere Eigenschaften der Beziehung unter den beteiligten Objekten nicht bekannt oder irrelevant sind.

Am Beispiel des Fahrwegs wird in Abb. 6.2 die Beziehung zwischen Fahrweg und seinen Teilkomponenten gezeigt. Dabei ist die Relation Aggregation durch eine leere Raute dargestellt, die auf das Aggregat zeigt. Die Relation Aggregation sagt hier nur, dass ein Objekt der Klasse *Fahrweg* als Teile Objekte der Klassen *Balise*, *Fahrwegelement* und *Gleisabschnitt* haben kann. Die Zahlenangaben 1 und 0..* zeigen, dass ein Objekt der Klasse *Fahrwegelement* zu einem Objekt der Klasse *Fahrweg* gehört und ein Objekt der Klasse *Fahrweg* kein oder einige Objekte der Klasse *Fahrwegelement* haben kann.

Die allgemeine Art der Zusammengehörigkeit, die durch Aggregation bezeichnet wird, charakterisiert keine exklusive Relation unter den beteiligten Objekten. Dabei können dieselben Objekte der Klassen *Balise*, *Fahrwegelement* und *Gleisabschnitt* zu anderen Zeitpunkten einem anderen Objekt der Klasse *Fahrweg* gehören.

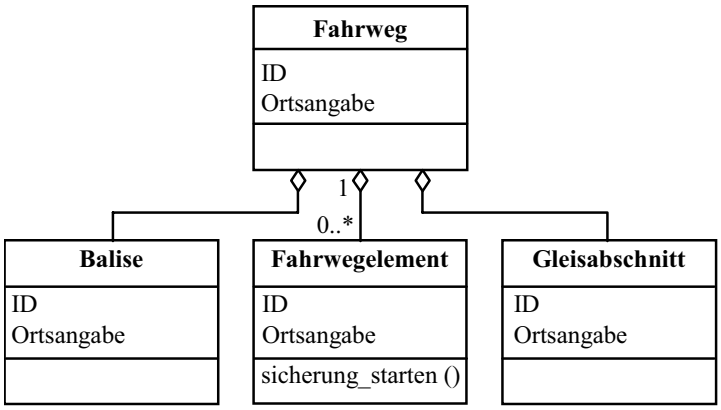


Abb. 6.2 Aggregation in einem Klassendiagramm

Damit haben ein bestimmtes Objekt fwg_1 der Klasse *Fahrweg* und ein bestimmtes Objekt bal_1 der Klasse *Balise* in den Mengen ihrer Relationen $\mathcal{R}_{fwg_1}$ und $\mathcal{R}_{bal_1}$ das Element fwg_1 *aggr* bal_1 . Die weiteren Elemente von $\mathcal{R}_{fwg_1}$ sind fwg_1 *aggr* fwe_1 , fwg_1 *aggr* gla_1 etc.

Da es sich bei der Aggregation um eine ganz allgemeine Form von Zusammengehörigkeit ohne weitere Abhängigkeiten handelt, lässt sich keine Aussage über die formalen Eigenschaften dieser Relation, wie Reflexivität, Symmetrie oder Transitivität treffen. Es

ist jedoch sinnvoll zu verlangen, dass sich bei der Verwendung dieser Relation unter den real existierenden Komponenten eines Systems kein Zyklus bilden darf, denn sonst würde das Aggregat wiederum als Teil des Teils definiert sein. Die folgende Regel schließt diesen Fall aus.

6.3.5 Regel

> Es seien K^1, ... , K^n Klassen und n bzw. j , k , ... , p , q , u und v natürliche Zahlen mit (jK^1 *aggr* kK^2), ... , (pK^{n-1} *aggr* qK^n). Dann gilt nicht (uK^n *aggr* vK^1).

Damit wird festgelegt, dass in einer Reihe von Objekten, die über Aggregation in einer Beziehung stehen, keines der Aggregate als Teil eines der Teile definiert werden kann.

6.3.6 Existenzabhängige Zusammengehörigkeit

Im Beispiel 6.3.2 wurde eine existenzabhängige Beziehung zwischen einem Bahnübergang und seinen Teilkomponenten gezeigt. Diese Art von Zusammengehörigkeit von Objekten wird Komposition genannt. Eine Komposition ist also eine besondere Form von Aggregation, in der eine existenz- und verhaltensabhängige sowie exklusive Zusammengehörigkeit von Objekten im Vordergrund steht. In den Klassendiagrammen wird eine Komposition wie in Abb. 6.3 mit gefüllter Raute bezeichnet, die auf das Ganze zeigt.

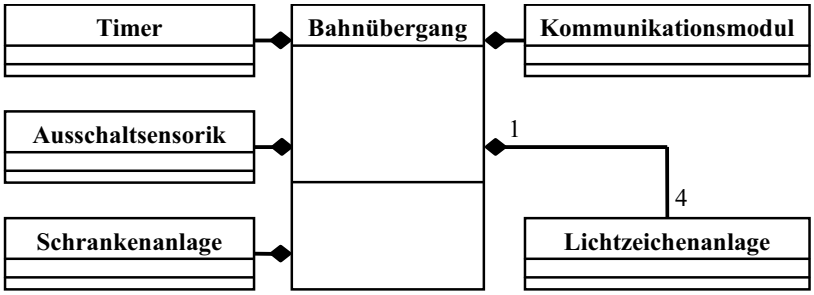


Abb. 6.3 Komposition in einem Klassendiagramm

In einem Klassendiagramm kann auch bei einer Komposition die Anzahl der beteiligten Objekte mit angegeben werden. Da Komposition eine exklusive Beziehung ist, ist die Kardinalität des Ganzen immer gleich eins. Die Zahlen in Abb. 6.3 bedeuten, dass zu einem Objekt der Klasse *Bahnübergang* vier Objekte der Klasse *Lichtzeichenanlage* gehören.

In Kapitel drei wurden die folgenden Charakteristika für die Relation Komposition festgelegt:

1. Unter dem Ganzen und seinen Teilen gibt es mindestens eine Assoziation.
2. Die Komponenten werden von dem Ganzen und von keinem anderen Objekt initialisiert und gelöscht.
3. Das Ganze wird von keiner Komponente erzeugt bzw. gelöscht.
4. Die Komponenten und das Ganze existieren immer zusammen.
5. Die Komponenten gehören in ihrer ganzen Lebensdauer ausschließlich zu einem einzigen Ganzen.

Ein Objekt $bü_1$ der Klasse *Bahnübergang* mit der Teilkomponente lzA_1 der Klasse *Lichtzeichenanlage* hat in der Menge seiner Aktionen die Aktion *lzA_einschalten*(lzA_1). Diese Aktion ist auch ein Element der Menge der Ereignisse von lzA_1 . Das bedeutet, dass es eine Assoziation zwischen $bü_1$ und lzA_1 gibt. Während $bü_1$ mit dem Ereignis *inBetriebNehmen*($bü_1$) und *außerBetriebNehmen*($bü_1$) initiiert und entfernt wird, führt $bü_1$ u. a. die Aktionen *inBetriebNehmen*(lzA_1) und *außerBetriebNehmen*(lzA_1) durch, die das Objekt lzA_1 initiieren bzw. löschen. Diese Aktionen gelten für lzA_1 als Initial- und Endereignis. Das Objekt $bü_1$ kann weder von dem Objekt lzA_1 noch von einem anderen Teilobjekt initiiert oder gelöscht werden. Schließlich gehört lzA_1 lediglich zu $bü_1$ und zu keinem weiteren Bahnübergang. Die weiteren Komponenten, wie die Schrankenanlage, die Ausschaltsensorik, das Kommunikationsmodul etc. werden in gleicher Art initiiert und gelöscht.

Die folgende Definition präzisiert die Eigenschaften der Existenzabhängigkeit und Exklusivität und charakterisiert die Relation Komposition. Die weiteren aufgezählten Eigenschaften lassen sich dann von dieser Definition ableiten.

6.3.7 Definition (Komposition)

Es seien K^p und K^q Klassen und *m* eine natürliche Zahl. Wenn jedes Objekt von K^p eine Komposition von genau *m* Objekten von K^q (K^p *comp* mK^q) ist, so gelten:

1. Bei den Initialtransitionen $\xrightarrow{e[b]/A}$ *default*() der Objekte der Klasse K^p gibt es in der Aktionenfolge $A = a_1 , ... , a_n$ genau *m* Aktionen der Form a_{init} (K^q), die Objekte der Klasse K^q erzeugen.
2. Bei den Finaltransitionen $z_i \xrightarrow{e[b]/A}$ *final*() der Objekte der Klasse K^p gibt es in der Aktionenfolge $A = a_1 , ... , a_n$ genau *m* Aktionen der Form a_{fin} (K^q), die Objekte der Klasse K^q löschen.

Wenn das Objekt $K^p_{O_i}$ eine Komposition der Objekte $K^q_{O_1}$, ... , $K^q_{O_m}$ ist, dann gehören die Elemente $K^p_{O_i}\ comp\ K^q_{O_1}$, ... , $K^p_{O_i}\ comp\ K^q_{O_m}$ der Menge der Relationen von $K^p_{O_i}$ und von $K^q_{O_1}$, ... , $K^q_{O_m}$.

3. $K^p_{O_i}\ comp\ K^q_{O_j} \wedge K^r_{O_k} comp\ K^q_{O_j} \Rightarrow p = r \wedge i = k.$

Der erste Teil dieser Definition setzt voraus, dass das Kompositionsobjekt das Teilobjekt erzeugt. Dies geschieht während der Initialtransition des Kompositionsobjekts, das heißt, während das Kompositionsobjekt selbst erzeugt wird, erzeugt es das Teilobjekt.

In dem zweiten Teil der Definition ist das Löschen des Teilobjekts definiert. Dieses geschieht während der Finaltransition des Kompositionsobjekts, das heißt, das Löschen des Teilobjekts ist eine Folge des Löschens des Kompositionsobjekts.

Das führt dazu, dass das Kompositionsobjekt und das Teilobjekt existenzabhängig sind, denn durch die genannten Voraussetzungen ist gewährleistet, dass das Erzeugen bzw. das Löschen des Kompositionsobjekts das Erzeugen bzw. das Löschen des Teilobjekts verursachen, das heißt, die beiden Objekte existieren gleichzeitig und sind dadurch existenzabhängig.

Die Existenz einer Aktion der Form $a_{fin}\ (K^{\ q})$ in einer Aktionenfolge des Kompositionsobjekts, die zugleich ein Ereignis für das Teilobjekt ist, zeigt, dass die Menge der Aktionen des Kompositionsobjekts und die Menge der Ereignisse des Teilobjekts mindestens ein gemeinsames Element haben, das heißt, $\mathcal{A}_{K^p} \cap \mathcal{E}_{K^q} \neq \varnothing$. Daraus folgt eine Assoziation unter den beteiligten Objekten, wie in UML [OMG03] bei der Relation Komposition vorausgesetzt wird. Die Existenz der gemeinsamen Elemente in den genannten Mengen zeigt zusätzlich noch die Verhaltensabhängigkeit zwischen dem Ganzen und seinen Teilen.

Der dritte Teil der Definition sagt, dass es sich um das gleiche Objekt handelt, wenn zwei Objekte als Kompositionsobjekt eines einzigen Teilobjekts definiert sind. Damit ist sichergestellt, dass die Beziehung zwischen den Teilobjekten und dem Kompositionsobjekt exklusiv ist und diese nur Teile eines einzigen Objekts sind.

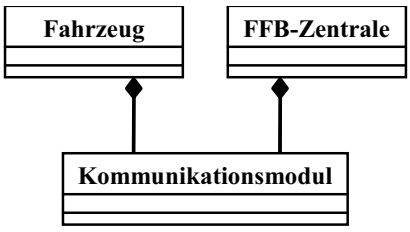


Abb. 6.4 Anwendung der Relation Komposition

Die Exklusivität der Relation Komposition bezieht sich auf die Objekte und nicht auf die Klassen. Daher bedeutet das in Abb. 6.4 dargestellte Klassendiagramm, dass jedes

Objekt der Klasse *Fahrzeug* bzw. jedes Objekt der Klasse *FFB-Zentrale* ein Objekt der Klasse *Kommunikationsmodul* hat und nicht jedes Objekt der Klasse *Kommunikationsmodul* ein Teil eines Objekts der Klasse *Fahrzeug* ist.

Wie bei der Aggregation ist es auch bei der Komposition sinnvoll zu verlangen, dass sich bei der Verwendung der Relation Komposition unter den real existierenden Komponenten eines Systems kein Zyklus bilden darf, denn sonst würde das Ganze wiederum als Teil des Teils definiert sein. Analog zur Regel 6.3.5 schließt die folgende Regel die Entstehung der Zyklen bei der Anwendung der Komposition aus.

6.3.8 Regel

Es seien $K^1, \ldots, K^n$ Klassen mit $(K^1 \; comp \; mK^2), \ldots, (K^{n-1} \; comp \; pK^n)$. Weiter seien n bzw. $m, \ldots, p$ und q natürliche Zahlen. Dann gilt nicht $(K^n \; comp \; qK^1)$.

Damit wird festgelegt, dass in einer Reihe von Objekten, die über Komposition in einer Beziehung stehen, keines der Kompositionsobjekte als Teil eines der Teilobjekte definiert werden kann.

Eine Folgerung aus dieser Regel ist, dass das Ganze weder von den Teilobjekten erzeugt noch gelöscht werden darf:

6.3.9 Regel

Es seien $K^1, \ldots, K^n$ Klassen mit $(K^1 \; comp \; mK^2), \ldots, (K^{n-1} \; comp \; pK^n)$. Weiter seien n bzw. $m, \ldots, p$ und q natürliche Zahlen.

Es gelten $a_{init}(K^1) \notin \mathcal{A}_{K^i}$ und $a_{fin}(K^1) \notin \mathcal{A}_{K^i}$ für alle $1 < i \leq n$.

Weiter lässt sich mit ähnlicher Argumentation wie bei der Assoziation zeigen, dass die Komposition eine nicht reflexive, nicht symmetrische und nicht transitive Relation ist.

1. Die Komposition ist nicht reflexiv, das heißt, $O_i \; comp \; O_i$ gilt nicht, anderenfalls muss O_i sich erzeugen und löschen können.
2. Die Komposition ist nicht symmetrisch, das heißt, $O_i \; comp \; O_j \Rightarrow O_j \; comp \; O_i$ gilt nicht, denn sonst wird ein Zyklus gebildet und das ist ein Wiederspruch zur Regel 6.3.8.
3. Die Komposition ist nicht transitiv, das heißt, für die Objekte O_i, O_j und O_k gilt nicht $O_i \; comp \; O_j \wedge O_j \; comp \; O_k \Rightarrow O_i \; comp \; O_k$, denn so würde O_k sowohl Teil von O_j als auch Teil von O_i sein. Das ist ein Wiederspruch zu dem dritten Teil der Definition 6.3.7.

6.4 Vererbung

Beim objektorientierten Ansatz zur Spezifikation von komplexen Systemen ist das Prinzip der Vererbung ein bedeutendes Konzept. Dabei handelt es sich um die Generalisierung der gemeinsamen Eigenschaften von Objekten in einer Superklasse, die dann von den Unterklassen spezialisiert werden. Bei einer objektorientierten Systemspezifikation führt der Einsatz dieses Prinzips zur Wiederverwendung und Redundanzvermeidung. In diesem Abschnitt wird das Prinzip der Vererbung aus der Sicht der Systemmodellierung betrachtet und dessen Einsatz am Beispiel der funkbasierten Bahnübergangssteuerung speziell für das Fahrwegelement und den eingleisigen Bahnübergang im FFB veranschaulicht. Ausgehend von diesem Beispiel wird für die Vererbung eine Definition präsentiert und ihre Bedeutung präzisiert.

6.4.1 Abstraktionsarten bei der objektorientierten Analyse

Bei der objektorientierten Analyse für die Modellierung eines Systems werden in der Regel zwei Arten von Abstraktionen vorgenommen. In einer Ebene werden die Objekte des Objektsystems nach Gemeinsamkeiten in Bezug auf ihre Eigenschaften und Funktionalitäten untersucht, um dadurch die Klassen bzw. Objektklassen zu definieren, welche die allgemeinen Eigenschaften von einzelnen Objekten präsentieren. In einem weiteren Schritt findet eine Untersuchung mit dem Ziel statt, die Analogien unter den Klassen zu abstrahieren und sie in einer übergeordneten Klasse darzustellen. Diese Art von Abstraktion umfasst die Darstellung von Übereinstimmungen und Differenzen zwischen verschiedenen Klassen. Dadurch können die bereits beschriebenen Eigenschaften einmal spezifiziert und mehrmals verwendet werden. Das führt zu der Redundanzvermeidung und einer sich dadurch ergebenden Fehlerreduzierung in der Systemspezifikation.

Das Abstrahieren von Gemeinsamkeiten der Klassen und die Darstellung dieser Eigenschaften in einer gesonderten Klasse wird *Generalisierung* genannt. Das Erben dieser Eigenschaften von den einzelnen Klassen ist dann die *Vererbung*. In [BE+94] wird die Generalisierung als eine Relation zwischen einer Klasse und einer oder mehreren verfeinerten Versionen davon definiert. Die Klasse, die verfeinert wird, heißt *Oberklasse*, *Superklasse* bzw. *Generalisierungsklasse* und jede verfeinerte Klasse heißt *Unterklasse* bzw. *Spezialisierungsklasse*. Daher ist eine Unterklasse eine spezialisierte Form der Oberklasse. Da diese Art von Vererbung ein essentieller Teil des objektorientierten Ansatzes ist, wird sie von allen objektorientierten Methoden wie Object Modeling Technique (OMT) oder UML unterstützt. Die objektbasierten Methoden hingegen wie z. B. Statemate [HP98] beschränken sich auf die Abstraktion unter den Objekten.

Bei der Vererbung werden die Attribute und die Aktionen, die für eine Gruppe von Unterklassen gelten, der Oberklasse zugewiesen und von den einzelnen Unterklassen gemeinsam genutzt. Jede Unterklasse erbt die Merkmale ihrer Oberklasse. Jedes Objekt einer Unterklasse ist auch ein Objekt der dazugehörigen Oberklasse. Weiter wird vor-

ausgesetzt, dass Generalisierung und Vererbung transitiv sind. Das bedeutet, dass eine Unterklasse nicht nur alle Merkmale ihrer direkten Oberklasse erbt, sondern auch alle ihrer Vorfahren und fügt diesen auch ihre eigenen individuellen Attribute und Aktionen hinzu.

Bei den objektorientierten Programmiersprachen wie C++ wird die Vererbung als eine gerichtete *ist-ein*-Beziehung definiert [Bre96]. Wenn in C++ eine Unterklasse von einer Oberklasse erbt, enthält jedes Objekt der Unterklasse ein Objekt der Oberklasse. Dabei wird das Objekt der Oberklasse noch vor der Erzeugung des Objekts der Unterklasse durch impliziten Aufruf des Oberklassenkonstruktors gebildet.

In [Bre96] wird dieser Aufruf anhand eines Bildes dargestellt, das für diese Arbeit in Abb. 6.5 an das Beispiel des Fahrwegelements und des eingleisigen und mehrgleisigen Bahnübergangs (MG-Bahnübergang) im FFB angepasst wurde.

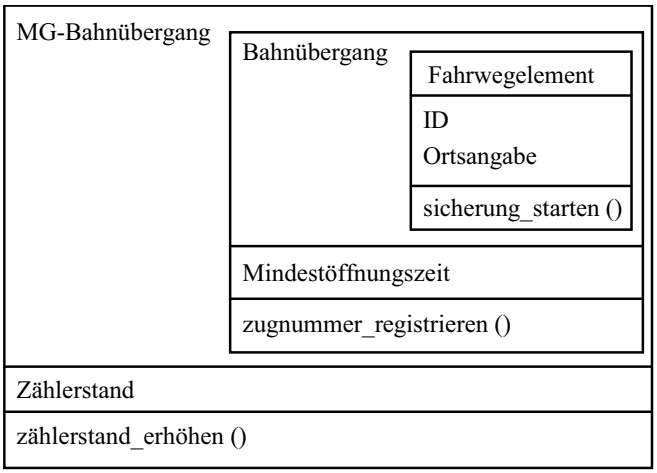


Abb. 6.5 Einschluss von Objekten der Oberklassen in ein Objekt der Unterklasse

Einige Sprachen wie Eiffel und C++ erlauben es, dass eine Unterklasse die Form einer Oberklasse erbt, während sie Aktionen selektiv von Vorfahren erbt und selektiv an Clients exportiert. Dies ist der Delegation gleichwertig, weil die Unterklasse nicht in jeder Hinsicht eine Form der Oberklasse ist und nicht mit ihr verwechselt wird [BE+94]. Da die objektorientierten Programmiersprachen Mechanismen bereitstellen sollen, die letztendlich die Vererbung von Quellcode ermöglichen, lassen sie zu, dass die Unterklasse nicht alle sondern nur einige der Eigenschaften ihrer Vorfahren erben. Daher machen sie die Eigenschaften der Oberklassen über Schlüsselwörter wie *public*, *private*, *protected* etc. für die Unterklassen zugänglich bzw. unzugänglich.

Eine Reduzierung der Eigenschaften der Oberklassen wird auch von einigen Autoren vorgesehen, die sich mit der objektorientierten Analyse und Design befassen. Diese Autoren sprechen explizit sogar von einer Vererbung zur Erweiterung und von einer Vererbung zur Einschränkung [Boo97]. In [Ehr99] wird unter Vererbung außer ist-ein-Relation noch als-ein-Relation diskutiert, die als Grundlage für die Vererbung zur Ein-

schränkung verwendet werden kann. Einige Autoren schließen unter den vererbbaren Eigenschaften von Objekten zusätzlich zu Attributen und Aktionen noch die Assoziationen ein [Bal99].

Das Gemeinsame an den unterschiedlichen Definitionen ist, dass die Vererbung

1. eine ist-ein-Relation darstellt,
2. eine reflexive, antisymmetrische und transitive Relation ist und
3. das Erben der Eigenschaften der Oberklassen durch die Unterklassen beinhaltet.

6.4.2 Vererbung von Eigenschaften

In Kapitel drei wurde das objektorientierte Modell zum FFB und zum technisch gesicherten eingleisigen Bahnübergang im FFB präsentiert. Dabei wurde erwähnt, dass die Unterklassen *Bahnübergang*, *Weiche* und *Schlüsselsperre* alle Attribute und Aktionen der Oberklasse *Fahrwegelement* erben. In diesem Abschnitt wird das allgemeine Verhalten von Fahrwegelementen diskutiert, das die Grundlage für die Festlegung des Verhaltens der Spezialisierungsklassen bildet.

In den Lastenheften zum FFB [ADt97] wird verlangt, dass bei jedem Fahrwegelement eine hohe Verfügbarkeit, sowie eine möglichst garantierte Ausfall-Fehleroffenbarung erreicht werden soll. Auftretende Störungen sollen sofort an die FFB-Zentrale gemeldet werden. Diese Anforderung setzt voraus, dass bei jedem Fahrwegelement eine Funktion definiert wird, die permanent den Ordnungszustand des Fahrwegelements überprüft und die FFB-Zentrale über einen eventuell vorliegenden Fehler in Kenntnis setzt. Weiter soll jedes Fahrwegelement auf eine Statusabfrage mit einer Statusmeldung antworten. Aus Sicherheitsgründen besteht eine weitere wichtige Anforderung darin, dass ein Fahrwegelement den Sicherungsvorgang nicht vor der vom System vorgesehenen Freigabe beenden darf.

Ebenso ist es aus betrieblichen Gründen vorstellbar, dass sich jedes zustandsvariable Fahrwegelement unter Umständen örtlich bedienen lässt, damit beim Auftreten von Unregelmäßigkeiten der Betrieb weiter geführt werden kann.

Diese Anforderungen führen dazu, dass sich unabhängig von der Art eines Fahrwegelements ein allgemeines Verhalten für zustandsvariable Fahrwegelemente definieren lässt. Um dieses allgemeine Verhalten darzustellen, ist es notwendig, allgemeine Zustände für das Verhalten eines Fahrwegelements zu definieren.

6.4.2.1 Verhalten eines Fahrwegelements

Für die Beschreibung der grundlegenden Zustände, die ein System annehmen kann, gibt es zahlreiche Betrachtungsweisen wie z. B. [Waw99]. Für die Anforderungen an ein

zustandsvariables Fahrwegelement im FFB wurde angenommen, dass sich ein Fahrwegelement zu jedem Zeitpunkt in einem der drei Zustände *Automatischer_Betrieb*, *Defekt* oder *Örtlich_bedienbar* befindet, wobei der Zustand *Automatischer_Betrieb* den regulären Zustand eines Fahrwegelements darstellt. *Örtlich_bedienbar* beschreibt, den Zustand, in dem Teile der Steuerungsaufgaben nicht automatisch von der Steuerungseinheit des Fahrwegelements, sondern vom Personal und über die vor Ort vorhandene Einrichtung ausgeführt werden können. Ein Fahrwegelement kann mit der Betätigung eines entsprechenden Schalters in den Zustand *Örtlich_bedienbar* gebracht oder aus diesem wieder in den Zustand *Automatischer_Betrieb* versetzt werden.

Defekt stellt den Zustand dar, in dem ein Sicherungsvorgang auf Grund eines vorliegenden Fehlers nicht begonnen bzw. nicht fortgesetzt werden kann.

Das Verhalten eines Fahrwegelements besteht nun aus Aktivitäten in diesen Zuständen und aus den Transitionen, die den Übergang zwischen diesen Zuständen bestimmen. Dieses allgemeine Verhalten bzw. der Lebenszyklus eines zustandsvariablen Fahrwegelements wurde bereits im Zustandsdiagramm in Abb. 3.5 dargestellt.

Mit dem Ereignis *inBetriebNehmen*() wird ein Objekt der Klasse *Fahrwegelement* erzeugt. Nach dem Erzeugen befindet sich das Objekt in dem Zustand *Automatischer_Betrieb*. Dieser Zustand besteht aus zwei parallelen Regionen. Dadurch wird gezeigt, dass die Aktivitäten in diesen Regionen gleichzeitig und parallel ausgeführt werden.

In einer dieser Regionen wird dargestellt, dass sich das Objekt ständig testet, denn in [ADt97] wird gefordert, dass jedes Fahrwegelement durch die Elementansteuerung mit einer eigenen Intelligenz ausgestattet ist, die dazu fähig ist, selbstständig und permanent den Status aller im Fahrwegelement enthaltenen Subsysteme zu überwachen.

Wird das Testen positiv abgeschlossen *Intakt*, so wird das Testen wiederholt. Wird beim Testen ein Fehler entdeckt *Defekt*, so verlässt das Objekt den Zustand *Automatischer_Betrieb* und geht in den Zustand *Defekt* über. Dabei wird die Aktion *defekt_melden*() ausgeführt. Diese Aktion erfüllt die Anforderung nach sofortiger Benachrichtigung der FFB-Zentrale beim Vorliegen einer Störung.

In der zweiten Region, in der der Zyklus eines Sicherungsvorgangs abgebildet ist, befindet sich ein Objekt der Klasse *Fahrwegelement* zuerst in dem Teilzustand *Frei*. In diesem Zustand reagiert das Fahrwegelement auf das Ereignis *element_anfordern*(), das von einem Objekt der Klasse *Fahrzeug* ausgelöst wird, mit der Aktion *quittieren*(). Damit quittiert das Fahrwegelement den Erhalt des Stellbefehls.

Ein Sicherungsvorgang kann gestartet werden, wenn die für das Fahrwegelement geltenden und die aus der Anforderung ableitbaren Voraussetzungen erfüllt sind. So startet z. B. ein Bahnübergang den Sicherungsvorgang, wenn der Einschaltzeitpunkt erreicht und die Mindestöffnungszeit abgelaufen ist. Die erste Bedingung wird aus dem Zeitpunkt abgeleitet, den das Objekt der Klasse *Fahrzeug* in seiner Elementanforderung

gesendet hat. Die Einhaltung der Mindestöffnungszeit ist jedoch von den Angaben unabhängig, die in der Elementanforderung enthalten sind. Sie ist eine für die Bahnübergänge spezifische Bedingung, die erfüllt sein muss, bevor erneut ein Sicherungsvorgang gestartet werden kann.

Bei einer Weiche gilt das Eintreffen einer Elementanforderung als die Startbedingung, denn entweder sie liegt bereits in der gewünschten Weichenlage und befindet sich bereits in dem gesicherten Zustand oder sie leitet ohne Verzögerung das Umstellen in die gewünschte Weichenlage ein, ohne weitere zeitliche oder betriebliche Bedingungen erfüllen zu müssen.

So verlässt das Fahrwegelement nach Darstellung in Abb. 3.6 den Zustand *Frei* und geht in den Zustand *Auftrag_bearbeitend* über, sobald die Bedingung *Startbedingung erfüllt* gilt. Dabei führt das Fahrwegelement die Aktion *sicherung_starten*() aus.

Kommt während der Bearbeitung des Sicherungsauftrags ein Fehler vor, so geht das Fahrwegelement in den Zustand *Defekt* über und führt dabei die Aktion *defekt_melden*() aus. Bei fehlerfreier Bearbeitung des Sicherungsauftrags bleibt das Fahrwegelement solange in dem Zustand *Auftrag_bearbeitend*, bis das Ereignis *freigeben*() eintritt. In diesem Fall geht das Fahrwegelement in den Zustand *Frei* über und führt die Aktion *sicherung_beenden*() aus.

Die Anforderungen nach fahrzeugseitiger Statusabfrage und elementseitiger Statusmeldung sind in dem Zustandsdiagramm der Klasse *Fahrwegelement* durch das Ereignis *status_abfragen*() und die Aktion *status_melden*() modelliert. Die Ausführung dieser Aktion ist mit dem *Deep-History-State* verbunden. Das bedeutet, dass das Objekt in dem Zustand *Automatischer_Betrieb* auf das Ereignis *status_abfragen*() mit der Aktion *status_melden*() reagiert und in dem Zustand fortsetzt, indem es beim Eintreten des Ereignisses *status_abfragen*() war.

Da das Ausführen dieser Aktion mit dem Deep-History-State verbunden ist, bleibt das Objekt der Klasse *Fahrwegelement* wieder in dem Teilzustand, in dem es sich beim Eintreten des Ereignisse *status_abfragen*() befand.

Das Ereignis *örtlich_schalten*(), das durch Betätigung eines entsprechenden Schalters vom Personal durchgeführt wird, führt ein Objekt der Klasse *Fahrwegelement* in den Zustand *Örtlich_bedienbar*. In diesem Zustand kann das Fahrwegelement örtlich vom Personal bedient werden. Dieser Zustand wird verlassen, wenn die örtliche Bedienung wieder ausgeschaltet wird. Diese Handlung wurde durch das Ereignis *autom_schalten*() dargestellt.

Falls noch möglich, reagiert ein Objekt der Klasse *Fahrwegelement* im Zustand *Defekt* mit der Aktion *status_melden*(*defekt*) auf eine Elementanforderung, die durch das Ereignis *element_anfordern*() abgebildet ist. Dadurch wird dem Fahrzeug mitgeteilt, dass das Fahrwegelement sich nicht sichern kann. Das Fahrzeug hat in diesem Fall vor dem Fahrwegelement zu halten. Dabei wird die Sicherung des Fahrwegelements vom Perso-

nal vor Ort durchgeführt. Der Zustand *Defekt* kann verlassen werden, wenn eines der Ereignisse *reparieren*() oder *außerBetriebNehmen*() eintritt.

6.4.2.2 Verhalten eines eingleisigen Bahnübergangs

Das Verhalten eines Fahrwegelements stellt dessen Lebenszyklus dar, der in Abb. 3.5 in Form eines Zustandsdiagramms angegeben wurde. Er gibt die möglichen Zustandsübergänge eines Objektes der Klasse *Fahrwegelement* wieder. Dieser Lebenszyklus wird von den Spezialisierungsklassen der Klasse *Fahrwegelement* geerbt und von denen um eigenes Verhalten erweitert und durch einige Verfeinerungstransformationen weiter spezialisiert.

Die Übertragung der Eigenschaften der Klasse *Fahrwegelement* auf die Klasse *Bahnübergang* und die speziellen Erweiterungen in der Klasse Bahnübergang sind in dem Zustandsdiagramm der Klasse *Bahnübergang* (Abb. 3.6) gezeigt. Ein Objekt der Klasse *Bahnübergang* befindet sich in seinem Lebenszyklus stets in einem der Zustände der Klasse *Fahrwegelement*, und die Übergänge zwischen den Zuständen eines Objekts der Klasse *Bahnübergang* entsprechen den Zustandsübergängen der Klasse *Fahrwegelement*.

In [Rum96] werden zwei unterschiedliche Lebenszyklen definiert, die durch den Typ- und den Klassenautomaten beschrieben werden. Der Typautomat beschreibt Lebenszyklen für alle Elemente einer Klasse und ihrer Spezialisierungsklassen, der Klassenautomat nur für die Instanzen, die direkt zu einer Klasse gehören. Dabei wird lediglich zugelassen, dass die Typautomaten vererbt werden. In [Rum96] dienen Typautomaten als Zusicherung von Verhaltenseigenschaften für die Instanzen einer Klasse, während die Klassenautomaten zur Implementierungsbeschreibung eingesetzt werden.

Durch diese Einschränkung lässt sich nur das Verhalten der Klassen mit abstrakten Objekten vererben, also Klassen, die selbst keine Instanzen haben und lediglich definiert werden, um einen Rahmen für das Verhalten der Spezialisierungsklassen festzulegen. Die Klasse *Fahrwegelement* entspricht solchen Klassen. Die Vererbung, die zwischen den Klassen *Bahnübergang* und *MG-Bahnübergang* stattfindet, ist jedoch eine Vererbung zwischen den konkreten Klassen, also zwischen Klassen, die selbst über Instanzen verfügen.

6.4.2.3 Vererbung der Struktur

Die Vererbung des Verhaltens einer Klasse impliziert auch die Vererbung ihrer Struktur, denn durch die Vererbung werden alle Eigenschaften und Funktionalitäten einer Klasse auf die Unterklassen übertragen. Existiert unter den Aktionen der Oberklasse eine Aktion, die in Zusammenhang mit einer Komponente dieser Klasse definiert wurde, so müssen die Spezialisierungsklassen auch eine entsprechende Komponente besit-

zen, die zur Ausführung dieser Aktion beiträgt. Aus dieser Überlegung folgt, dass die Komponenten einer Oberklasse mit vererbt werden.

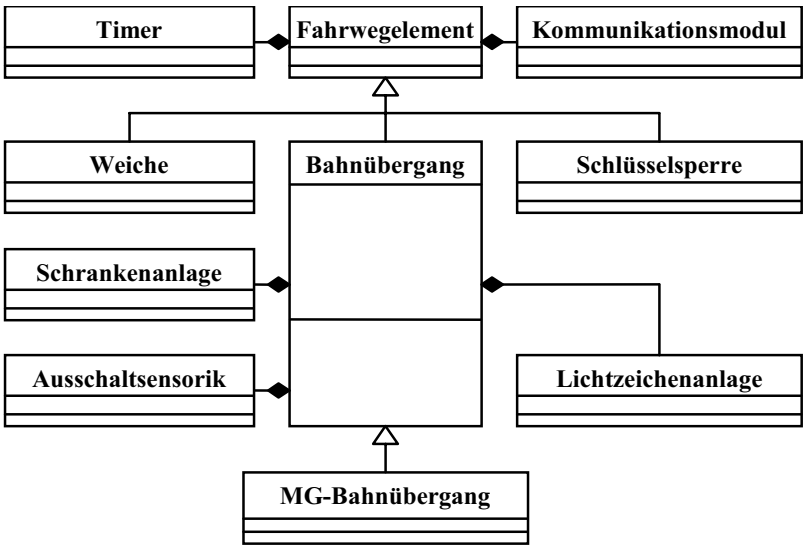


Abb. 6.6 Vererbung in einem Klassendiagramm

Nach [ADt97] kommuniziert jedes Fahrwegelement, das systemgesteuert in das FFB-System eingebunden ist, über ein Kommunikationsmodul mit dem Fahrzeug oder der FFB-Zentrale. Das bedeutet für eine objektorientierte Modellierung des Fahrwegelements, dass eine Objektklasse *Kommunikationsmodul* als Teil der Objektklasse *Fahrwegelement* definiert werden kann, die auch als Teil der Spezialisierungsklassen *Bahnübergang*, *Weiche* und *Schlüsselsperre* gilt. Ebenfalls wird hier eine weitere Objektklasse *Timer* als ein Teil der Objektklasse *Fahrwegelement* definiert, die zur Modellierung der zeitlichen Aspekte in dem objektorientierten Modell dient. In Abb. 6.6 ist das für die Objektklasse *Fahrwegelement* resultierende Klassendiagramm abgebildet.

Nach dieser Darstellung besitzt ein Objekt der Klasse *Fahrwegelement* ein Objekt der Klasse *Kommunikationsmodul* und ein Objekt der Klasse *Timer*. Dementsprechend haben auch die Spezialisierungsklassen *Bahnübergang*, *Weiche* und *Schlüsselsperre* jeweils ein Objekt der Klasse *Kommunikationsmodul* und ein Objekt der Klasse *Timer*.

Die Spezialisierungsklassen fügen ihre eigenen speziellen Teilkomponenten hinzu. So hat dann ein Objekt der Klasse *Bahnübergang* ein Objekt der Klasse *Schrankenanlage*, ein Objekt der Klasse *Lichtzeichenanlage* und ein Objekt der Klasse *Ausschaltsensorik*. Ein Objekt der Klasse *MG-Bahnübergang* (mehrgleisiger Bahnübergang) erbt alle der genannten Komponenten. Dadurch werden diese Komponenten nur einmal definiert und mehrmals wiederverwendet.

Ausgehend von den vorangegangenen Betrachtungen wird die Relation Vererbung, wie folgt, definiert. Diese Definition enthält eine Übertragung der Struktur und des dynamischen Verhaltens.

6.4.3 Definition (Vererbung)

> Es sei K die Menge aller Klassen. Dann ist Vererbung eine partielle Zuordnung *Inherit*: $K \rightarrow K$ mit *Inherit*: $K^q \mapsto K^p$, welche die Struktur und das Verhalten der Klasse K^q auf die Klasse K^p überträgt. K^q wird die Generalisierungsklasse und K^p die Spezialisierungsklasse genannt. Die Bezeichnung (K^p *inherit* K^q) zeigt die Relation zwischen den Klassen K^p und K^q.

Die Relation Vererbung wird zwischen den Klassen definiert und gilt für jedes Objekt der Spezialisierungsklasse, das heißt, K^p *inherit* K^q ist ein Element der Menge der Relationen der Klasse K^p und gehört nicht zu der Menge der Relationen der Klasse K^q. Jedes Objekt $K^p_{O_i}$ der Klasse K^p hat ein Element $K^p_{O_i}$ *inherit* K^q in der Menge seiner Relationen.

6.4.4 Erläuterung der Zuordnung Inherit:

> Für $K^p = (\mathcal{A}ttr_{K^p}$, $\mathcal{E}_{K^p}$, $\mathcal{B}_{K^p}$, $\mathcal{A}_{K^p}$, $\mathcal{M}_{K^p}$, $\mathcal{R}_{K^p})$ und $K^q = (\mathcal{A}ttr_{K^q}$, $\mathcal{E}_{K^q}$, $\mathcal{B}_{K^q}$, $\mathcal{A}_{K^q}$, $\mathcal{M}_{K^q}$, $\mathcal{R}_{K^q})$ mit *Inherit*: $K^q \mapsto K^p$ bzw. (K^p *inherit* K^q) wird jeder Komponente des Tupels von K^p die entsprechende Komponente des Tupels von K^q zugeordnet.

wie im Folgenden gezeigt wird, impliziert *inherit* eine Abbildung, die die Elemente des ersten Tupels auf die Elemente des zweiten Tupels abbildet.

In der Menge der Klassen ist die Zuordnung *Inherit* partiell, das heißt, es besteht nicht unter allen sondern nur unter einigen Klassen die Beziehung Vererbung.

Bezüglich der Eigenschaften der beteiligten Klassen ist *inherit* total. In den weiteren Abschnitten wird *inherit* in Bezug auf die einzelnen Mengen der Attribute, Ereignisse, Bedingungen, Aktionen, Relationen und in Bezug auf die Zustandsmaschinen der Generalisierungs- und Spezialisierungsklasse erläutert.

6.4.4.1 Abbildung der Attribute:

> Aus *Inherit*: $K^q \mapsto K^p$ folgt *inherit*: $\mathcal{A}ttr_{K^q} \rightarrow \mathcal{A}ttr_{K^p}$, wobei diese Abbildung eine Einbettung von $\mathcal{A}ttr_{K^q}$ in $\mathcal{A}ttr_{K^p}$ ist.
>
> Durch die Einbettung, die auch Inklusion genannt wird, ist jedes Attribut in K^q auch in K^p vorhanden.

Bei der Vererbung unter den Klassen *Fahrwegelement*, *Bahnübergang* und *MG-Bahnübergang* sind *ID*, t_{akt} , *Frei*, *Defekt* etc. Attribute der Klasse *Fahrwegelement*, die von der Menge der Attribute dieser Klasse auf die Menge der Attribute der Klasse

Bahnübergang und von da auf die Menge der Attribute der Klasse *MG-Bahnübergang* übertragen werden. Da die Abbildung *inherit* hier total ist, werden alle Attribute der Generalisierungsklasse auf die Spezialisierungsklasse übertragen.

Die Abbildung der einzelnen Attribute erfolgt durch das folgende Schema:

inherit: $ID \mapsto ID$, *inherit*: $t_{akt} \mapsto t_{akt}$ etc.

Zusätzlich zu den Attributen, die von der Klasse *Fahrwegelement* übernommen werden, hat die Menge der Attribute der Klasse *Bahnübergang* die speziellen Elemente t_G , t_l , t_{gs} etc. Die sämtlichen Attribute der Klasse *Bahnübergang* werden dann auf die Menge der Attribute der Klasse *MG-Bahnübergang* übertragen. In der Menge der Attribute dieser Klasse gibt es zusätzlich zu den genannten Attributen noch das Element *Einschaltzähler* als spezielles Attribut für die Objekte der Klasse *MG-Bahnübergang*.

6.4.4.2 Abbildung der Ereignisse:

> Aus *Inherit*: $K^q \mapsto K^p$ folgt *inherit*: $\mathcal{E}_{K^q} \rightarrow \mathcal{E}_{K^p}$. Diese Abbildung ist injektiv, das heißt, für jedes Ereignis in K^q gibt es ein entsprechendes Ereignis in K^p.

Bei der Vererbung unter den Klassen *Fahrwegelement*, *Bahnübergang* und *MG-Bahnübergang* sind *element_anfordern*(), *status_abfragen*(), *örtlich_schalten*(), *reparieren*() etc. Ereignisse der Klasse *Fahrwegelement*, die von der Menge der Ereignisse dieser Klasse auf die Menge der Ereignisse der Klasse *Bahnübergang* abgebildet werden. Da diese Abbildung total ist, werden alle Ereignisse der Generalisierungsklasse auf die Spezialisierungsklasse übertragen.

Die Abbildung der einzelnen Ereignisse erfolgt durch das folgende Schema:

inherit: *element_anfordern*() $\mapsto$ *element_anfordern*(*bü*), *inherit*: *status_abfragen*() $\mapsto$ *status_abfragen*() etc.

Zusätzlich zu den von der Klasse *Fahrwegelement* gerbten Ereignissen hat die Menge der Ereignisse der Klasse *Bahnübergang* die speziellen Elemente *licht_melden*(*rot*), *licht_melden*(*gelb*), *schA_melden*(*geschlossen*), *schA_melden*(*geöffnet*) etc. Die sämtlichen Ereignisse der Klasse *Bahnübergang* werden dann auf die Menge der Ereignisse der Klasse *MG-Bahnübergang* abgebildet.

6.4.4.3 Abbildung der Bedingungen:

> Aus *Inherit*: $K^q \mapsto K^p$ folgt *inherit*: $\mathcal{B}_{K^q} \rightarrow \mathcal{B}_{K^p}$, das heißt, für jede Bedingung in K^q gibt es eine entsprechende Bedingung in K^p.

Bei der Vererbung unter den Klassen *Fahrwegelement*, *Bahnübergang* und *MG-Bahnübergang* ist z. B. *Startbedingungen_erfüllt*() eine Bedingung in der Klasse *Fahrwegelement*, die von der Menge der Bedingungen dieser Klasse auf die Menge der Be-

dingungen der Klasse *Bahnübergang* und auf die Menge der Bedingungen der Klasse *MG-Bahnübergang* übertragen wird. Diese Bedingung entspricht der Bedingung $t_e \leq t_{akt}$ UND $t_{mo} \leq t_{akt}$ UND *zugnummer_registriert*. Diese gilt ihrerseits als Startbedingung für den Sicherungsvorgang bei einem Objekt der Klasse *Bahnübergang*. Da die Abbildung *inherit* hier total ist, werden alle Bedingungen der Generalisierungsklasse auf die Spezialisierungsklasse übertragen.

Die Abbildung der genannten Bedingung erfolgt durch das folgende Schema:

inherit: *Startbedingungen_erfüllt*() $\mapsto$ $t_e \leq t_{akt}$ UND $t_{mo} \leq t_{akt}$ UND *zugnummer_registriert*.

Zusätzlich zu den Bedingungen, die von der Klasse *Fahrwegelement* übernommen werden, hat die Menge der Bedingungen der Klasse *Bahnübergang* die speziellen Elemente $t_G \leq t_{akt}$, $t_l \leq t_{akt}$, $t_{gs} \leq t_{akt}$ etc. Alle Bedingungen der Klasse *Bahnübergang* werden dann auf die Menge der Bedingungen der Klasse *MG-Bahnübergang* abgebildet.

6.4.4.4 Abbildung der Aktionen:

> Aus *Inherit*: $K^q \mapsto K^p$ folgt *inherit*: $\mathcal{A}_{K^q} \rightarrow \mathcal{A}_{K^p}$. Diese Abbildung ist injektiv, das heißt, für jede Aktion in K^q gibt es eine entsprechende Aktion in K^p.

Bei der Vererbung unter den Klassen *Fahrwegelement*, *Bahnübergang* und *MG-Bahnüber-gang* sind *quittieren*(), *status_melden*(), *sicherung_starten*() etc. Aktionen der Klasse *Fahrwegelement*, die von der Menge der Aktionen dieser Klasse auf die Menge der Aktionen der Klasse *Bahnübergang* und auf die Menge der Aktionen der Klasse *MG-Bahnübergang* übertragen werden. Während die Aktionen *quittieren*() und *status_melden*() den gleichnamigen Aktionen in der Menge der Aktionen der Klasse *Bahnübergang* entsprechen, wird die Aktion *sicherung_starten*() auf die Aktion *lzA_einschalten*() in der Menge der Aktionen der Klasse *Bahnübergang* abgebildet, denn bei einem Bahnübergang wird der Sicherungsvorgang mit dem Einschalten der Lichtzeichenanlage begonnen. Da die Abbildung *inherit* hier total ist, werden alle Aktionen der Generalisierungsklasse auf die Spezialisierungsklasse übertragen.

Die Abbildung der einzelnen Aktionen erfolgt durch das folgende Schema:

inherit: *quittieren*() $\mapsto$ *quittieren*(), *inherit*: *status_melden*() $\mapsto$ *status_melden*(), *inherit*: *sicherung_starten*() $\mapsto$ *lzA_einschalten*() etc.

Zusätzlich zu den von der Klasse *Fahrwegelement* übernommenen Aktionen besitzt die Menge der Aktionen der Klasse *Bahnübergang* die speziellen Elemente *lzA_umschalten*(), *lzA_ausschalten*(), *schA_einschalten*(), *schA_ausschalten*() etc. Alle Aktionen der Klasse *Bahnübergang* werden auf die Menge der Aktionen der Klasse *MG-Bahnübergang* übertragen.

6.4.4.5 Abbildung der Zustandsmaschinen:

Aus *Inherit*: $K^q \mapsto K^p$ folgt *inherit*: $\mathcal{M}_{K^q} \rightarrow \mathcal{M}_{K^p}$. Dabei werden alle Zustände und alle Transitionen übertragen.

Die Eigenschaften dieser Abbildungen sind in der folgenden Definition gegeben:

6.4.4.5.1 Definition (Abbildung von Zustandsmaschinen)

Es seien $\mathcal{M}_{K^p} = (\mathcal{Z}_{K^p}, \mathcal{T}_{K^p}, default()_{K^p}, final()_{K^p})$, $\mathcal{M}_{K^q} = (\mathcal{Z}_{K^q}, \mathcal{T}_{K^q}, default()_{K^q}, final()_{K^q})$ die Zustandsmaschinen von K^p und K^q mit *Inherit*: $K^q \mapsto K^p$. Bei der Abbildung *inherit*: $\mathcal{M}_{K^q} \rightarrow \mathcal{M}_{K^p}$ gelten:

1. $default()_{K^q} = default()_{K^p}$,
2. $final()_{K^q} = final()_{K^p}$,
3. $\mathcal{Z}_{K^q} \subseteq \mathcal{Z}_{K^p}$,
4. Für jede Transition $t' \in \mathcal{T}_{K^q}$ mit t': $z_i \rightarrow z_j$ gibt es eine Transition $t \in \mathcal{T}_{K^p}$ mit der Eigenschaft t: $z_i \rightarrow z_j$.

Durch die ersten zwei Bedingungen wird festgelegt, dass die Initial- bzw. Finalzustände der Objekte der beteiligten Klassen identisch sind, das heißt, die Objekte der Generalisierungsklasse und die der Spezialisierungsklasse beginnen bzw. beenden ihren Lebenszyklus in dem bzw. aus dem gleichen Zustand. Bei der Vererbung unter den Klassen *Fahrwegelement* und *Bahnübergang* (Abb. 3.5, 3.6) beginnen die Objekte der Klasse *Fahrwegelement* und die Objekte der Klasse *Bahnübergang* ihren Lebenszyklus in dem Zustand *Automatischer_Betrieb*. Sie beenden ihren Lebenszyklus aus dem Teilzustand *Frei* in dem Zustand *Automatischer_Betrieb*.

Die dritte Bedingung verlangt, dass alle Zustände der Generalisierungsklasse in der Menge der Zustände der Spezialisierungsklasse enthalten sind. Damit wird sichergestellt, dass die Objekte der Spezialisierungsklasse alle für die Generalisierungsklasse definierten Zustände einnehmen. Bei der Vererbung unter den Klassen *Fahrwegelement* und *Bahnübergang* sind die Zustände *Automatischer_Betrieb*, *Defekt*, *Örtlich_Bedienbar*, *Testend*, *Frei* und *Auftrag_Bearbeitend* Zustände der Objekte der Klasse *Fahrwegelement*, die auch für die Objekte der Klasse *Bahnübergang* gelten.

Durch die vierte Bedingung wird gewährleistet, dass die in der Generalisierungsklasse definierten Zustandsübergänge auch bei den Objekten der Spezialisierungsklasse stattfinden können.

Damit sich jedoch die Objekte der Spezialisierungsklasse immer in einem der Zustände der Generalisierungsklasse befinden und damit die Zustandsübergänge bei Objekten der

Spezialisierungsklasse gemäß Transitionen in der Generalisierungsklasse erfolgen, wird die folgende Regel eingeführt:

6.4.4.5.2 Regel

Bezüglich $\mathcal{Z}_{K^p}$, $\mathcal{T}_{K^p}$, $\mathcal{Z}_{K^q}$ und $\mathcal{T}_{K^q}$ aus der Definition 6.4.4.5.1 gelten:

1. $\forall\, z \in (\mathcal{Z}_{K^p} \setminus \mathcal{Z}_{K^q}) \; \exists\, z' \in \mathcal{Z}_{K^q} \,.\, z \in \mathit{in}(z')$ und
2. Wenn bei einer Transition $t \in \mathcal{T}_{K^p}$ mit t: $z_i \rightarrow z_j$ ein Zustand $z'_i \in \mathcal{Z}_{K^q}$ verlassen und ein $z'_j \in \mathcal{Z}_{K^q}$ aktiviert wird, dann existiert eine Transition $t' \in \mathcal{T}_{K^q}$, durch die z'_i verlassen und z'_j aktiviert wird.

In dieser Regel wird zuerst festgelegt, dass die erweiterten Zustände der Spezialisierungsklasse nur als Teilzustände der in der Generalisierungsklasse definierten Zustände definiert sein dürfen. Damit wird sichergestellt, dass die Objekte der Spezialisierungsklasse in ihrem Lebenszyklus sich immer in einem der Zustände der Generalisierungsklasse befinden und keine weiteren Zustände einnehmen, die nicht in der Generalisierungsklasse definiert sind.

Durch die zweite Bedingung werden in der Spezialisierungsklasse die Zustandsübergänge zwischen den von der Generalisierungsklasse geerbten Zustände gemäß ihrer Anordnung in der Generalisierungsklasse übertragen

Diese Bedingungen gewährleisten eine Übernahme des Verhaltens der Objekte der Generalisierungsklasse durch die Objekte der Spezialisierungsklasse.

Bei der Vererbung unter den Klassen *Fahrwegelement* und *Bahnübergang* sind die erweiterten Zustände *Gelb_erwartend*, *Rot_erwartend*, *Gesichert*, *Belegt* etc. Teilzustände des Zustands *Auftrag_Bearbeitend*, der bereits als ein Zustand bei der Klasse *Fahrwegelement* definiert worden ist. Die Zustandsübergänge von *Rot_erwartend* nach *Defekt* bzw. *Geschlossen_erwartend* nach *Defekt* in der Klasse *Bahnübergang* entsprechen dem Zustandsübergang von *Auftrag-Bearbeitend* nach *Defekt*, der in der Klasse *Fahrwegelement* definiert worden ist. Durch diese Transitionen werden auch bei den Objekten der Klasse *Bahnübergang* die Zustände *Auftrag_Bearbeitend* und *Automatischer_Betrieb* verlassen und der Zustand *Defekt* aktiviert.

Durch das Einführen der Teilzustände in die bereits definierten Zustände lässt sich das allgemeine Verhalten weiter spezifizieren ohne den gesamten Rahmen für das dynamische Verhalten zu verlassen. Darüber hinaus kann das Verhalten durch das Hinzufügen weiterer Aktionen in die Aktionenfolgen der Generalisierungsklasse erweitert werden. Dabei können in jedem Zustand z einige spezielle Aktionen in die Aktionenfolgen $en(z)$, $act(z)$, $do(z)$ oder $ex(z)$ hinzugefügt werden.

Die Erweiterung einer Aktionenfolge besteht darin, dass die neue Aktion an die gewünschte Stelle eingefügt wird. In der Klasse *Fahrwegelement* steht in dem Teilzustand

Frei die Aktionenfolge *quittieren*(), die als Reaktion auf das Ereignis *element_anfordern*() definiert wurde. In der Klasse *Bahnübergang* wird diese Aktionenfolge mit zwei weiteren Aktionen t_e*_setzen*() und *zugnummer_registrieren*() erweitert.

6.4.4.6 Abbildung der Relationen:

Aus *Inherit*: $K^q \mapsto K^p$ folgt *inherit*: $\mathcal{R}_{K^q} \rightarrow \mathcal{R}_{K^p}$. Diese Abbildung ist injektiv, das heißt, für jede Relation in K^q gibt es eine entsprechende Relation in K^p.

Die Abbildung der einzelnen Relationen erfolgt durch das folgende Schema:

inherit: $(nK^q \; rel \; mK^r) \mapsto (nK^p \; rel \; mK^r)$ für alle *r* mit $(nK^q \; rel \; mK^r) \in \mathcal{R}_{K^q}$.

Bei der Vererbung unter den Klassen *Fahrwegelement* und *Bahnübergang* werden durch die Abbildung *inherit*: $\mathcal{R}_{Fahrwegelement} \rightarrow \mathcal{R}_{Bahnübergang}$ die Elemente (*Fahrwegelement comp Timer*), (*Fahrwegelement comp Kommunikationsmodul*), (*Fahrzeug asso *Fahrwegelement*), die Relationen der Klasse *Fahrwegelement* sind, auf (*Bahnübergang comp Timer*), (*Bahnübergang comp Kommunikationsmodul*), (*Fahrzeug asso *Bahnübergang*) abgebildet, die Relationen der Klasse *Bahnübergang* sind.

Da die Abbildung *inherit* total ist, werden alle Relationen der Generalisierungsklasse auf die Spezialisierungsklasse übertragen.

Zusätzlich zu den Relationen, die von der Klasse *Fahrwegelement* übernommen werden, hat die Menge der Relationen der Klasse *Bahnübergang* die speziellen Elemente in Form von (*Bahnübergang comp Schrankenanlage*), (*Bahnübergang comp Lichtzeichenanlage*) und (*Bahnübergang comp Ausschaltsensorik*). Alle Relationen der Klasse *Bahnübergang* werden nach demselben Schema auf die Menge der Relationen der Klasse *MG-Bahnübergang* übertragen.

6.4.5 Folgerung:

1. Vererbung ist eine reflexive Relation, das heißt, für jede Klasse K^p gilt (K^p *inherit* K^p). In diesem Fall ist *inherit* bei der Abbildung der Attribute, Ereignisse, Bedingungen, Aktionen, Zustandsmaschine und Relationen die identische Abbildung, das heißt, bei jeder Abbildung wird jedes Element auf sich selbst abgebildet.
2. Vererbung ist weder eine symmetrische, noch eine asymmetrische noch eine antisymmetrische Relation.
   2.1. Aus (K^p *inherit* K^q) folgt nicht (K^q *inherit* K^p). Durch das Beispiel mit den Klassen *Fahrwegelement* und *Bahnübergang* ist klar, dass jedes Objekt der Klasse *Bahnübergang* ein Objekt der Klasse *Fahrwegelement* ist jedoch nicht jedes Objekt der Klasse *Fahrwegelement* ist ein Objekt der

Klasse *Bahnübergang*. Damit ist Vererbung keine symmetrische Relation.

2.2. Aus (K^p *inherit* K^q) folgt nicht, dass (K^q *inherit* K^p) nicht gilt. Für diesen Fall ist (K^p *inherit* K^p) ein Gegenbeispiel. Damit ist Vererbung keine asymmetrische Relation.

2.3. Aus (K^p *inherit* K^q) und (K^q *inherit* K^p) folgt nicht $K^p = K^q$. Die Abbildungen bezüglich der Ereignisse, Bedingungen, Aktionen etc. sind keine identischen Abbildungen. Daher können es bezüglich K^p und K^q Unterschiede hinsichtlich der genannten Eigenschaften geben. Daher folgt nicht $K^p = K^q$. Damit ist Vererbung keine antisymmetrische Relation.

3. Vererbung ist eine transitive Relation, das bedeutet, aus (K^p *inherit* K^q) und (K^q *inherit* K^r) folgt (K^p *inherit* K^r).

3.1. Aus (K^p *inherit* K^q) folgt *inherit*: $\mathcal{R}_{K^q} \rightarrow \mathcal{R}_{K^p}$. Daher gilt für alle K^t mit (nK^q *rel* mK^t) $\in \mathcal{R}_{K^q}$ die Eigenschaft (nK^p *rel* mK^t) $\in \mathcal{R}_{K^p}$. Daraus folgt (K^p *inherit* K^r) $\in \mathcal{R}_{K^p}$, denn es gilt (K^q *inherit* K^r).

6.4.6 Mehrfache Vererbung

Erbt eine Unterklasse die Eigenschaften mehrerer Oberklassen, so heißt diese Art von Vererbung mehrfache Vererbung. Einige objektorientierte Methoden lassen mehrfache Vererbung zu [Rum96] und einige andere lehnen diese Art von Vererbung ab [DH97].

Abhängig von den Zielen, die bei der Modellierung eines Systems verfolgt werden, können eine oder alle Arten von Vererbung benutzt werden. In der vorliegenden Arbeit wird aufgrund der Vererbung der Struktur und des dynamischen Verhaltens keine mehrfache Vererbung zugelassen. Hier wird eine Relation gemäß Definition 6.4.3 Vererbung genannt und für die Spezifikationen der Systemkomponenten benutzt. Da es sich hierbei um die Vererbung des Verhaltens handelt, würde eine mehrfache Vererbung bedeuten, dass sich die Objekte der Spezialisierungsklasse gleichzeitig in den Zuständen mehrerer Generalisierungsklassen befinden sollten. Eine mehrfache Vererbung erschwert das Nachvollziehen der vererbbaren Eigenschaften und damit die Validierung und die Verifikation der Systemdefinition.

Mit den hier diskutierten Relationen lassen sich die Systemstruktur und die Beziehungen zwischen den Komponenten eines Systems spezifizieren. Zusammen mit den Definitionen aus dem vorangegangenen Kapitel für die Zustandsdiagramme und die Sequenzdiagramme stehen nun grafische und formal fundierte Beschreibungsmittel zur Verfügung, die eine präzise Beschreibung von drei verschiedenen Sichten eines Systems ermöglichen. Die Klassendiagramme und die darin enthaltenen Relationen ermög-

lichen eine Spezifikation der Systemstruktur. Mit Hilfe von Zustandsdiagrammen kann das interne Verhalten von Systemkomponenten beschrieben werden und die Interaktionen zwischen den Systemkomponenten können durch die Sequenzdiagramme spezifiziert werden. Die Frage der Konsistenz und Konsistenzbedingungen unter diesen drei Sichten lässt sich auf Basis der eingeführten Definitionen beantworten.

6.5 Konsistenzbedingungen für Teile der Systemspezifikation

Die in diesem Kapitel eingeführten Definitionen für die Relationen zur Beschreibung des Systemaufbaus basieren auf den dynamischen Eigenschaften der Systemkomponenten. Eine Assoziation ist gemäß Definition 6.2.1 gegeben, wenn ein Objekt eine Aktion bei einem anderen Objekt auslöst, das heißt, wenn eine Interaktion zwischen den beteiligten Objekten stattfindet. Eine Komposition ist gemäß Definition 6.3.7 gegeben, wenn die Teilobjekte unmittelbar mit dem Erzeugen des Ganzen erzeugt und unmittelbar mit seinem Löschen aus dem System entfernt werden.

Weiter besteht das als Prozess bzw. Interaktion zwischen den Systemkomponenten und in Sequenzdiagrammen beschriebene Systemverhalten gemäß Abschnitt 5.11 aus Aktionen, die Objekte ausführen. Diese Aktionen gehören zu dem Verhalten von Objekten, das anhand der Zustandsdiagramme spezifiziert wird.

Diese Eigenschaften implizieren die folgenden Bedingungen unter Klassen-, Zustands-, und Sequenzdiagrammen, die hier Konsistenzbedingungen unter den Spezifikationsteilen genannt werden. Damit ist das objektorientierte Modell bzw. die aus den genannten Diagrammarten bestehende Systemanforderungsspezifikation in sich konsistent, wenn:

1. für jede Assoziation in den Klassendiagrammen mindestens ein Sequenzdiagramm definiert wurde, in dem Objekte der an der Assoziation beteiligten Klassen vorkommen,
2. für jede Komposition in den Klassendiagrammen die Existenzabhängigkeit der beteiligten Objekte in den entsprechenden Zustandsdiagrammen gegeben ist und
3. die in den Sequenzdiagrammen definierten Interaktionen durch das in den Zustandsdiagrammen definierte lokale Verhalten erfüllt werden.

Die Überprüfung des in den Zustandsdiagrammen definierten Verhaltens wird im folgenden Kapitel ausgeführt. Diese Überprüfung findet anhand einer formalen Verifikation mit Beweischarakter statt. Die angewendete formale Verifikationsmethode ist das Model-Checking, das es ermöglicht, ein System in allen möglichen Kombinationen seiner Zustände gegenüber Prüfbedingungen zu überprüfen. Dabei können die Prüfbedingungen bestimmte Sicherheitsanforderungen oder Anforderungen an die Funktionalität des Systems sein, die z. B. in Sequenzdiagrammen spezifiziert worden sind. Durch die formale Überprüfung des lokalen Verhaltens, das in den Zustandsdiagrammen model-

liert wird, gegenüber den in den Sequenzdiagrammen modellierten Interaktionen können die Korrektheit und die Konsistenz dieser Modellteile gezeigt werden.

7 Verifikation

In diesem Kapitel wird gezeigt, wie das objektorientierte Modell auf Basis der Technik Model-Checking gegenüber Prüfbedingungen formal verifiziert werden kann. Im Mittelpunkt dieses Kapitels steht der Weg von dem objektorientierten Modell bis zur formalen Verifikation. Zuerst werden hier die formale Verifikation und die Technik des Model-Checking vorgestellt. In einem weiteren Teil dieses Kapitels wird der Schritt zur Vorbereitung eines objektorientierten Modells für die Durchführung der formalen Verifikation ausführlich demonstriert. Dabei wird die Transformation des dynamischen Systemverhaltens, das in Form von Zustandsdiagrammen der UML entworfen wurde, in eine formal verifizierbare Form erläutert. Danach wird auf die Logiksprache eingegangen, die zur Formulierung der Prüfbedingungen verwendet wird. Zum Schluss wird auf die Probleme mit der Anwendung der Technik Model-Checking eingegangen.

7.1 Verifikation (allgemein)

Mit dem Begriff Verifikation werden Aktivitäten bezeichnet, die sich mit der Frage beschäftigen, ob eine Anforderungsbeschreibung bestimmte Kriterien, wie etwa Vollständigkeit und Wiederspruchsfreiheit, erfüllt [Par98]. In [Lig02] wird mit Verifikation (allgemein) die Überprüfung bezeichnet, ob die Ergebnisse einer Entwicklungsphase die Anforderungen zu Beginn der Phase erfüllen.

Bei der Entwicklung von Eisenbahnsicherungsanlagen ist die Erstellung der Anforderungsbeschreibung in Form von Lastenheften die erste Entwicklungsphase. In dieser Phase sind die Anforderungen aus den Normen, Vorschriften, Eisenbahngesetzten und -verordnungen die Anforderungen, die in den Lastenheften als Ergebnis dieser Phase erwähnt bzw. berücksichtigt werden müssen.

Mit der Vollständigkeit geht es darum, ob die Systemanforderungsbeschreibung z. B. zurückgestellte Bemerkungen oder Verweise auf nicht existierende Komponenten etc. enthält [Par98]. Weiter ist es wichtig, ob es Hinweise auf alle relevanten Gesetzte und Vorschriften gibt. Bei den natürlich sprachlich verfassten Lastenheften finden derartige Untersuchungen manuell statt. Jedoch eine Konsistenzüberprüfung bezüglich der funktionalen Systemanforderungen ist nahezu unmöglich.

Eine Verifikation der Systemanforderungen kann in unterschiedlicher Art und Weise durchgeführt werden. Jede Verifikationsart kann nur auf Basis einer geeigneten Anforderungsbeschreibungsart erfolgen. In den nächsten Abschnitten werden die formale Verifikation und die dafür erforderlichen Anforderungsbeschreibungsarten diskutiert.

7.2 Formale Verifikation

Eine formale Verifikation ist eine Verifikation, in der die mathematischen Mittel zur Durchführung der Verifikation verwendet werden. Mithilfe einer formalen Verifikation können funktionale Systemanforderungen derart überprüft werden, dass die Verifikation den Charakter einer Beweisführung im mathematischen Sinne hat. Ist die formale Verifikation der Systemanforderungsspezifikation mit Erfolg abgeschlossen, so ist der Systemüberprüfer in der Lage, zu behaupten, dass alle funktionalen Anforderungen in der Anforderungsbeschreibung berücksichtigt wurden und konsistent sind.

Ein weiterer Vorteil der formalen Verifikationen besteht darin, dass sie rechnergestützt und automatisch durchgeführt werden. Dadurch können alle möglichen Kombinationen von Vorkommnissen bei den Systemabläufen vollständig untersucht werden. Derartige Untersuchungen sind bei komplexen Systemen nicht manuell durchführbar.

Eine der gängigen und rechnergestützten formalen Verifikationsmethoden mit mathematischem Beweischarakter ist das Model-Checking.

7.3 Model-Checking

Model-Checking ist ein algorithmisches Verfahren, das nachweist, ob in einem gegebenen strukturierten Modell eine Anforderung erfüllt ist. Dieses Verfahren wurde 1981 von Clarke und Emerson in [CE81a] und von Quielle und Sifakis in [QS81] vorgestellt. Dabei wird das strukturierte Modell in Form von in Abschnitt 7.5 eingeführten Kripke-

Strukturen bzw. in Form eines Automaten und die Anforderung bzw. die Prüfbedingung in Form einer temporallogischen Formel niedergeschrieben. In [BB+01] werden diese Technik, die Tools und die angewendeten temporallogischen Sprachen ausführlich diskutiert.

Die Werkzeuge, welche die Technik des Model-Checking unterstützen, unterscheiden sich einerseits in der angewendeten Beschreibungsart des Systemmodells und andererseits in der angewendeten temporallogischen Sprache. Bei UPPALL [LPW98] wird das zu untersuchende System in Form von zeitbehafteten Automaten (Timed Automata) modelliert und die Prüfbedingungen werden in Computation Tree Logic (CTL) oder Linear Temporal Logic (LTL) formuliert. SPIN [Hol97] verwendet für die Systembeschreibung die protokollbasierte Sprache PROMELA und für die Formulierung der Prüfbedingungen LTL.

Der Symbolic Model Vrifier (SMV) setzt eine Systembeschreibung in Form von Kripke-Strukturen (Abschnitt 7.5) voraus. Dieser Model-Checker kann sowohl die Formeln in CTL als auch in LTL interpretieren. Bei den Prüfbedingungen in LTL ist allerdings eine automatenbasierte Systembeschreibung notwendig.

In der vorliegenden Arbeit wird das Model-Ckecking mit SMV als formale Verifikation gewählt. Diese Entscheidung wurde aus dem praktischen Grund getroffen, denn von den Ingenieuren Anwendungsprogramme gewünscht werden, die auf dem Betriebssystem Windows ausführbar sind. Zur Zeit ist SMV der einzige Model-Checker mit dieser Eigenschaft.

In Kapitel drei wurde gezeigt, dass die Zustandsdiagramme der UML durch entry-, do-, exit-actions und durch hierarchische und parallele Zustände sowie durch History-States und Deep-History-States Möglichkeiten bieten, das dynamische Verhalten der Systemkomponenten präzise und adäquat zu beschreiben. Diese lassen sich, wie im Folgenden dargestellt wird, in Kripke-Strukturen als Eingangssprache von SMV transformieren.

SMV [McM96] wurde 1996 an der Carnegie-Mellon Universität von Ken McMillan entwickelt und ist unter [http://www-2.cs.cmu.edu/~modelcheck/smv.html] frei verfügbar.

7.3.1 Model-Checking mit SMV

SMV überprüft die Korrektheit von Prüfbedingungen in einer Kripke-Struktur, auf die in Abschnitt 7.5 eingegangen wird. Erfüllt die Kripke-Struktur eine Prüfbedingung, so wird sie nach dem Abschluss der Verifikation mit „true“ gekennzeichnet. Für die Prüfbedingungen, die mit „false“ bezeichnet werden, wird von SMV ein Gegenbeispiel in Form eines Pfades angezeigt, in dem die Prüfbedingung verletzt wird. Dieses Gegenbeispiel kann wieder zur Korrektur des Modells verwendet werden.

Für die formale Verifikation der funktionalen Anforderungen im UML-Modell, muss einerseits das dynamische Verhalten der Systemkomponenten, das in Form von Zustandsdiagrammen der UML vorliegt, in Kripke-Strukturen transformiert werden, und andererseits die Prüfbedingungen in CTL formuliert werden. Abb. 7.1 zeigt diesen Vorgang.

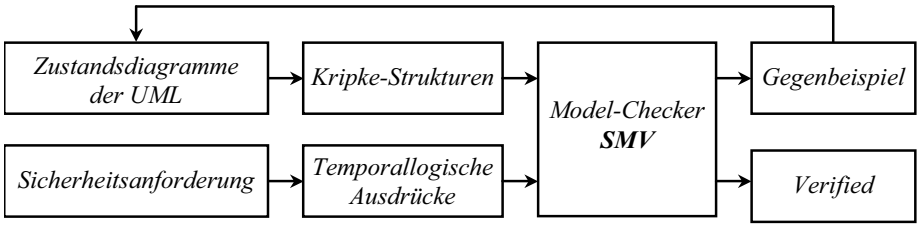


Abb. 7.1 Ablauf der formalen Verifikation von Zustandsdiagrammen der UML mit SMV

In dem nächsten Abschnitt wird zunächst der Schritt von den Zustandsdiagrammen der UML nach Kripke-Strukturen erläutert.

7.4 Transformation der Zustandsdiagramme

Die Umschreibung von Zustandsdiagrammen bzw. Statecharts in einige formal verifizierbare Formen hat es bereits gegeben. Die formale Verifikation in z. B. [DK01] basiert auch auf den Statecharts, die in geeignete Automaten umgeschrieben worden sind. Dabei werden die Statecharts gegenüber Live Sequence Charts (LSCs) [DH01, BD+04], die in lineare Automaten transformiert werden, überprüft. Die dabei zugrunde liegenden Diagramme sind die Statecharts [Har87], die in dem Werkzeug Statemate [HP98] implementiert wurden und sich in einigen Punkten von den UML-Zustandsdiagrammen unterscheiden.

Hier werden auf Basis der Beschreibungen in Kapitel fünf Transformationsschemata für die Umschreibung jeder Eintragung in einem Zustandsdiagramm präsentiert. Mit Hilfe dieser Schemata lassen sich die Zustandsdiagramme der UML in geeignete Kripke-Strukturen übertragen. Dadurch können sie von einem Model-Checker formal verifiziert werden.

Eine Kripke-Struktur wird über eine Menge von atomaren Aussagen definiert. Beispiele für atomare Aussagen sind „*Lichtzeichenanlage ist eingeschaltet*“, „*Gelb leuchtet*“, „*Rot leuchtet nicht*“, „*Mindestgrünzeit ist abgelaufen*“ etc. In den Kripke-Strukturen, die aus den Zustandsdiagrammen der UML nach dem hier vorgestellten Verfahren entstehen, gelten noch „*einschalten*“, „*defekt_melden*“ etc. als atomare Aussagen. Während diese Ausdrücke in einem Zustandsdiagramm die Durchführung einer Aktion zeigen, bezeichnen diese in einer Kripke-Struktur den Zustand, in dem die Aktion bereits ausgeführt und abgeschlossen worden ist.

Für eine Kripke-Struktur gibt es in der Literatur unterschiedliche Definitionen. In [CGP00] wird sie, wie folgt, definiert:

7.5 Definition (Kripke-Struktur)

Sei V eine Menge atomarer Aussagen. Eine Kripke-Struktur KS über V ist durch (S, S_0, R, L) definiert. Dabei sind:

1. S eine nichtleere endliche Menge von (Kripke-) Zuständen,
2. $S_0 \subseteq S$ die Menge der Anfangszustände,
3. $R \subseteq S \times S$ die Transitionsrelation, die für alle Elemente von S definiert ist und
4. $L : S \rightarrow \mathcal{P}(V)$ eine Abbildung (Beschriftungsfunktion), die jedem (Kripke-) Zustand eine Menge der gültigen atomaren Aussagen zuordnet.

Durch diese Definition wird eine Kripke-Struktur als eine Menge von Kripke-Zuständen S definiert, in der einige Kripke-Zustände, also Elemente von S_0, als Anfangszustand gekennzeichnet werden. Für alle Kripke-Zustände sind die Beziehungen zu den anderen Kripke-Zuständen in der Menge R festgelegt. Schließlich ist der Informationsinhalt jedes Kripke-Zustands durch die Funktion L gegeben.

In Abb. 7.2 ist die grafische Darstellung einer Kripke-Struktur gegeben. Diese Kripke-Struktur ist über $V = \{ p, q \}$ die Menge der atomaren Aussagen p und q definiert. Dabei sind $S = \{ s_0, s_1, s_2, s_3 \}$ die Menge der Kripke-Zustände und $S_0 = \{ s_0 \}$ die Menge der Anfangszustände. Die Transitionsrelation R ist die Menge, die aus den Paaren (s_0, s_1), (s_0, s_3), (s_1, s_2), (s_1, s_3), (s_2, s_3) und (s_3, s_2) besteht. Die Menge der in jedem Kripke-Zustand gültigen Aussagen ist neben jedem Kripke-Zustand notiert. So liefert L z. B. für s_0 die Menge $\{ p \}$ und für s_3 die Menge $\{ p, q \}$.

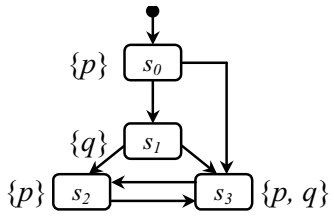


Abb. 7.2 Grafische Darstellung einer Kripke-Struktur

In [HR00] ist eine ähnliche Definition gegeben, die sich lediglich darin unterscheidet, dass für die Kripke-Strukturen kein Anfangszustand angegeben wird.

In einer Kripke-Struktur kann unter jedem Zustand die Belegung der Variablen gespeichert werden. Die Variablen sind die Attribute des Objektes, deren Typen und Initialwerte bereits in dem entsprechenden Klassendiagramm festgelegt worden sind und de-

ren Werte durch Aktionen des Objektes geändert werden. Bei den Aktivitäten eines Objekts kann die Ausführung jeder Aktion als ein Zustand und die Auswirkung der Aktion auf die Attributwerte als eine atomare Aussage aufgefasst werden. Damit gibt es für jede Aktion des Objekts einen Zustand in der Kripke-Struktur, der die Ausführung der entsprechenden Aktion des Objekts bezeichnet. Für ein Objekt der Klasse Bahnübergang entsprechen die Aktionen t_e*_setzen*(), t_G*_setzen*(), *lzA_einschalten*() etc. jeweils einem Zustand in der entsprechenden Kripke-Struktur. In diesen Zuständen sind die Ausdrücke t_e = 15:30:30, t_G = 15:30:33, *lzA_eingeschaltet* = *false* etc. atomare Aussagen, die in diesen Zuständen gelten.

Diese Strukturen können durch einen Model-Checker verifiziert werden. Dabei besteht die Verifikation aus der Überprüfung einer Zusammensetzung atomarer Aussagen in temporallogischer Formulierung, deren Gültigkeit in den Zuständen der Kripke-Struktur untersucht wird.

Obwohl bei einer Kripke-Struktur von Zuständen und Transitionen gesprochen wird, unterscheiden sich diese von den Zuständen und Transitionen in einem Zustandsdiagramm der UML. Während ein Zustand in einem Zustandsdiagramm die Ausführung bestimmter Aktivitäten zeigt, bezeichnet ein Zustand in einer Kripke-Struktur die aktuellen Werte der Attribute nach der Ausführung von bestimmten Aktivitäten. Ebenso zeigt eine Transition in einem Zustandsdiagramm das Beenden bestimmter Aktivitäten und das Beginnen von anderen Aktivitäten, während eine Transition in einer Kripke-Struktur nur auf die nächsten geltenden Attributwerte nach der Ausführung von Aktionen zeigt. Um eine eventuelle Verwechslung zwischen einem Zustand in einer Kripke-Struktur und einem Zustand in einem Zustandsdiagramm der UML zu vermeiden, wird hier der Begriff Kripke-Zustand für einen Zustand in einer Kripke-Struktur verwendet. Ebenso werden die Transitionen in einer Kripke-Struktur als Verknüpfung zwischen zwei Kripke-Zuständen bezeichnet.

7.6 Transformation des Modells

In dem objektorientierten Modell wurde für jede Klasse bzw. für jedes Objekt ein Zustandsdiagramm definiert, das das interne Verhalten des entsprechenden Objekts darstellt. Bei der Verifikation des Modells besteht der erste Schritt aus der Transformation des in den Notationen der UML erstellten Modells in ein formal verifizierbares Modell. Dieses Modell umfasst eine Abbildung der Zustandsdiagramme, die das dynamische Verhalten der einzelnen Objekte wiedergeben, in Form von Kripke-Strukturen und die Kommunikationen unter den Objekten. Das verifizierbare Modell kann dann gegenüber einzelnen Anforderungen überprüft werden. Dadurch lässt sich untersuchen, ob die Objekte in ihrem Lebenszyklus alle gewünschten Anforderungen erfüllen oder es Fälle gibt, in denen eine Anforderung verletzt wird.

Die Abbildung des dynamischen Verhaltens der Systemkomponenten erfolgt dadurch, dass das Zustandsdiagramm jedes einzelnen Objekts in eine separate Kripke-Struktur transformiert wird. In der Kripke-Struktur werden alle Ausführungsschritte des Objekts bei allen möglichen Abläufen abgebildet. Dadurch kann die Auswirkung jeder einzelnen Aktion des Objekts auf die Attributwerte in jedem Ablauf festgelegt und untersucht werden. Die Überprüfung des Verhaltens des gesamten Systems erfolgt dann über eine Untersuchung aller Objekte und ihrer Kommunikationen miteinander gegenüber gestellten Anforderungen.

In Kapitel drei wurde gezeigt, dass die Kommunikationen unter den Objekten über Aktionen bzw. Ereignisse stattfinden. Ein Objekt führt als Reaktion auf ein Ereignis eine Aktion durch oder nimmt einen Zustand ein. Das Ereignis selbst ist das Ergebnis einer Aktion, die von einem anderen Objekt ausgeführt wurde. Die Kommunikation der Objekte kann aber auch von Bedingungen abhängig sein, die mit Attributwerten der kommunizierenden Objekte zusammenhängen. Bei dem verifizierbaren Modell besteht die Modellierung der Kommunikationen unter den Systemkomponenten aus der Abbildung der genannten Abhängigkeiten zwischen Aktionen eines Objekts und Aktionen bzw. Zustandswechseln eines anderen Objekts.

In dem nächsten Abschnitt wird die Transformation der Zustandsdiagramme in geeignete Kripke-Strukturen präsentiert.

7.7 Transformation von UML-Zustandsdiagrammen in Kripke-Strukturen

In Kapitel drei wurde gezeigt, dass die Zustandsdiagramme der UML die Möglichkeit bieten, die Aktionen eines Objekts in unterschiedlichen Anordnungen zu notieren. Diese Möglichkeit ist dadurch gegeben, dass in einem Zustandsdiagramm eine Menge von eventuell ineinander geschachtelten oder parallelen Zuständen und Transitionen zwischen diesen Zuständen entworfen werden können. Darüber hinaus kann in einem Zustand z anhand der Bezeichnungen $en(z)$, $act(z)$, $ex(z)$ etc. bestimmt werden, ob die Aktionen des Objektes beim Eintritt in z, währen sich das Objekt in z befindet oder beim Verlassen von z ausgeführt werden. In Kripke-Strukturen sind solche Möglichkeiten nicht gegeben. Daher werden hier geeignete Schemata für die Transformation von Notationen der Zustandsdiagramme der UML definiert, die dann bei der Transformation eines Zustandsdiagramms der UML teilweise oder alle verwendet werden. Diese Schemata umfassen eine Abbildung von Zuständen und den darin enthaltenen Aktionenfolgen sowie eine Abbildung von Transitionen und ihren Notationen auf die Kripke-Struktur.

Die Ereignisse bzw. die Bedingungen, die im Kontext der Ausführungsbedingungen einzelner Aktionen oder der Austrittsbedingungen der Zustände definiert werden, wer-

den in dem verifizierbaren Modell, das aus Kripke-Strukturen und deren Kommunikationen besteht, durch die Kommunikation über gemeinsame Variablen implementiert.

7.7.1 Transformation von Aktionenfolgen

Aktionen und Aktionenfolgen können in einem Zustandsdiagramm der UML in Form von entry-actions, do-actions etc. bzw. im Kontext der Transitionen vorkommen. In Kapitel fünf wurde erläutert, dass die in einem Zustand definierten Aktionenfolgen in verschiedenen Reihenfolgen ausgeführt werden.

Die Transformation einer Aktionenfolge ist die Umwandlung ihrer einzelnen Aktionen in Kripke-Zustände und die Festlegung der Auswirkung der einzelnen Aktionen auf die Attributwerte in jedem Ausführungsschritt.

In Abschnitt 5.1.2.1 wurde festgestellt, dass in einer Aktionenfolge jede Aktion erst ausgeführt werden kann, wenn ihre Ausführungsbedingung erfüllt ist. Die Aktionenfolge wird unvollständig ausgeführt, wenn die Ausführungsbedingung einer ihrer Aktionen nicht erfüllt ist. Daher ergeben sich für eine Aktionenfolge unterschiedliche Ausführungsmöglichkeiten. Da in der entsprechenden Kripke-Struktur alle möglichen Ausführungen abgebildet werden sollen, ergeben sich bei jeder Aktion zwei Möglichkeiten, wenn die darauf folgende Aktion eine Ausführungsbedingung hat, die nicht erfüllt sein kann. Die eine Möglichkeit besteht darin, dass die Aktionenfolge an dieser Stelle endet, weil die genannte Ausführungsbedingung nicht erfüllt ist und die zweite Möglichkeit ist die Ausführung der darauf folgenden Aktion.

Zusätzlich zu Ausführungsbedingungen einzelner Aktionen kann die gesamte Aktionenfolge selbst eine Ausführungsbedingung haben, die nicht erfüllt sein kann. Hierbei ergeben sich wiederum zwei Fälle für die Ausführung bzw. für keine Ausführung der Aktionenfolge.

Das folgende Schema umfasst die genannten Fälle.

7.7.2 Transformationsschema 1 (TS1):

Es sei $A = a_1 , \ldots , a_n$ eine Aktionenfolge mit der Ausführungsbedingung (e_0 , b_0), deren Ausführung die Werte der Attribute in $\mathcal{A}trr$ verändert. Weiter sei (e_i , b_i) die Ausführungsbedingung von a_i mit $1 \leq i \leq n$. Die entsprechende Kripke-Struktur KS_A wird durch (S, S_0 , R, L) mit folgenden Eigenschaften definiert:

1. $S = \{ a_1 , \ldots , a_n \} \cup \{ a_0 , a_{n+1} \}$ mit $a_0 = a_{n+1} = a_\varepsilon$. Dabei bezeichnen a_0 bzw. a_{n+1} die Fälle vor bzw. nach der Ausführung von A ,
2. $S_0 = \{ a_0 \}$,

3. $R = \{(a_i , a_{i+1}) \mid 0 \le i \le n \} \cup \{(a_{i-1} , a_{n+1}) \mid (e_i , b_i) \neq (e_\varepsilon , b_\varepsilon) \text{ für } 1 \le i \le n \}$ und
4. $L : S \rightarrow \mathcal{P}(\mathcal{A}trr)$ liefert in jedem Zustand die aktuellen Werte der Attribute in $\mathcal{A}trr$, die sich aus der Ausführung jeder Aktion ergeben.

Die erste Zeile dieses Schemas wandelt alle Aktionen der Aktionenfolge in die gleichnamigen Kripke-Zustände um und fügt denen a_0 bzw. a_{n+1} hinzu. Dabei bezeichnen a_0 den Fall vor der Ausführung der Aktionenfolge und a_{n+1} den Abschluss der Aktionenfolge, der vollständig oder nicht vollständig erfolgen kann. Damit ist die Menge der Anfangszustände dieser Kripke-Struktur gleich $\{ a_0 \}$.

Die Transitionsrelation dieser Kripke-Struktur besteht aus der Vereinigung zweier Mengen. In der ersten Menge kommen Transitionen vor, welche die sequenzielle Ausführung der Aktionen gemäß ihrer Reihenfolge in der gegebenen Aktionenfolge abbilden. Durch Elemente dieser Menge wird die vollständige Ausführung der Aktionenfolge abgebildet, in der nach der Ausführung jeder Aktion die Ausführungsbedingung der darauf folgenden Aktion erfüllt ist. Die zweite Menge beinhaltet Transitionen, welche die Aktionen, deren darauf folgende Aktion eine Ausführungsbedingung (e_i , b_i) hat, die sich von (e_ε , b_ε) unterscheidet, und dadurch eventuell nicht erfüllt sein kann, als letzte Aktion der Aktionenfolge mit dem Kripke-Zustand a_{n+1} verbinden und damit den Abschluss der Aktionenfolge abbilden. Die Elemente dieser Menge ermöglichen die Abbildung der Fälle in die Kripke-Struktur, in denen eine nicht vollständige Ausführung der Aktionenfolge vorkommt. In Abb. 7.3 ist eine grafische Darstellung dieser Transformation gegeben.

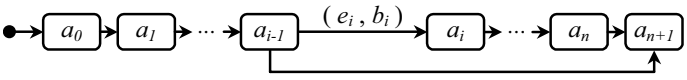


Abb. 7.3 Abbildung einer Aktionenfolge in eine Kripke-Struktur

Wie in Kapitel fünf erläutert wurde, können die Aktionenfolgen in einem Zustand *z* unter anderem im Kontext der Aktivitäten von *do*(*z*) vorkommen. Während in sonstigen Fällen eine Aktionenfolge mit der letzten ausführbaren Aktion beendet wird, wird sie in dem Fall von *do*(*z*) solange wiederholt, bis die Austrittsbedingung von *z* erfüllt ist und damit der Zustand *z* verlassen wird. Daher ergibt sich bei der Transformation von Aktionenfolgen in die Kripke-Strukturen der folgende Sonderfall für die Aktionenfolgen als *do*(*z*).

7.7.3 Transformationsschema 2 (TS2):

Ist $A = a_1 , \ldots , a_n$ als *do*(*z*) eines Zustands *z* definiert, so wird in TS1 bei der Menge *S* auf das Hinzufügen von a_{n+1} verzichtet, da die do-actions nicht abge-

schlossen, sondern immer wieder wiederholt werden, solange das Objekt z nicht verlassen hat. Damit wird für die Aktionenfolge A als do-actions die Kripke-Struktur KS_A durch (S, S_0, R, L) mit folgenden Eigenschaften definiert:

1. $S = \{ a_1, \ldots, a_n \} \cup \{ a_0 \}$ mit $a_0 = a_\varepsilon$. Dabei bezeichnet a_0 den Fall vor der Ausführung von A,
2. $S_0 = \{ a_0 \}$,
3. $R = \{(a_i, a_{i+1}) \mid 0 \le i < n \} \cup \{(a_n, a_0)\} \cup \{(a_{i-1}, a_0) \mid (e_i, b_i) \neq (e_\varepsilon, b_\varepsilon)$ für alle $1 \le i \le n \}$ und
4. $L : S \rightarrow \mathcal{P}(\mathcal{A}trr)$ liefert in jedem Zustand die aktuellen Werte der Attribute in $\mathcal{A}trr$, die sich aus der Ausführung jeder Aktion ergeben.

Bei der Menge R beinhalten die letzten zwei Mengen der Vereinigung Transitionen, welche die Kripke-Zustände, die sich aus den letzten ausführbaren Aktionen der Aktionenfolge ergeben, mit dem Anfangszustand der Kripke-Struktur verbinden, und damit die Wiederholungen der *do*(z) abbilden.

Eine grafische Darstellung dieser Transformation ist in Abb. 7.4 dargestellt.

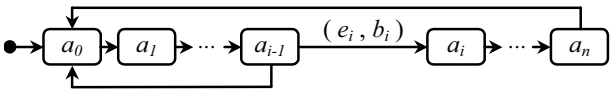


Abb. 7.4 Abbildung einer Aktionenfolge als do-actions in eine Kripke-Struktur

7.7.4 Transformation einer Folge von Aktionenfolgen

In einem Zustand z können zwei oder mehrere Aktionenfolgen vorkommen, die von dem Objekt sequenziell durchgeführt werden. Beispiele dafür sind *en*(z) und *act*(z). In z kann mit der Ausführung von *act*(z) begonnen werden, wenn die Ausführung von *en*(z) beendet wird und die Austrittsbedingung von z nicht erfüllt ist.

Bei der Transformation solcher Fälle auf eine Kripke-Struktur, die sich aus den Kripke-Strukturen der einzelnen Aktionenfolgen bildet, besteht die Zustandsmenge der gesamten Kripke-Struktur aus der Vereinigung der Zustandsmengen der einzelnen Kripke-Strukturen. Dadurch sind alle beteiligten Aktionen in der gesamten Kripke-Struktur vertreten. Der Anfangszustand der gesamten Kripke-Struktur ist dann der Anfangszustand der Kripke-Struktur der ersten Aktionenfolge.

In der Transitionsrelation der gesamten Kripke-Struktur müssen alle Übergänge gemäß ihrem Vorkommen in der jeweiligen Aktionenfolge abgebildet werden. Daher müssen alle Elemente der Transitionsrelationen der beteiligten Kripke-Strukturen in der Transitionsrelation der gesamten Kripke-Struktur aufgenommen werden.

Da nach der Ausführung der ersten Aktionenfolge mit der Ausführung der zweiten Aktionenfolge begonnen wird, wenn die Ausführungsbedingung der zweiten Aktionenfolge erfüllt ist, muss es außer Übergängen, die in den beteiligten Kripke-Strukturen gelten, noch einen Übergang von dem letzten Kripke-Zustand der ersten Kripke-Struktur in den ersten Kripke-Zustand der zweiten Kripke-Struktur geben.

Das Transformationsschema 3 präsentiert eine formale Beschreibung der Kripke-Struktur, die sich aus dem sequenziellen Ablauf zweier Aktionenfolgen ergibt.

7.7.5 Transformationsschema 3 (TS3):

Es seien $A' = a'_1, \ldots, a'_n$ und $A'' = a''_1, \ldots, a''_m$ Aktionenfolgen, deren Ausführung die Werte der Attribute in $\mathcal{A}trr$ verändert. Weiter seinen $KS_{A'} = (S', S'_0, R', L')$ und $KS_{A''} = (S'', S''_0, R'', L'')$ die entsprechenden Kripke-Strukturen für A' und A''.

Für die Aktionenfolge $A'; A''$, in der A' und A'' sequenziell ablaufen, ergibt sich die Kripke-Struktur KS durch (S, S_0, R, L) mit folgenden Eigenschaften:

1. $S = S' \cup S''$,
2. $S_0 = \{ a'_0 \}$,
3. $R = R' \cup R'' \cup \{(a'_{n+1}, a''_0)\}$ und
4. Es gilt $L = L' \cup L''$ und $L : S \rightarrow \mathcal{P}(\mathcal{A}trr)$ liefert in jedem Zustand die aktuellen Werte der Attribute in $\mathcal{A}trr$, die sich aus der Ausführung jeder Aktion ergeben.

In Abb. 7.5 wird die Kripke-Struktur, die sich nach dem genannten Schema ergibt, grafisch dargestellt. Dabei sind (e'_i, b'_i) und (e''_j, b''_j) Ausführungsbedingungen von a'_i und a''_j mit $1 \le i \le n$ und $1 \le j \le m$.

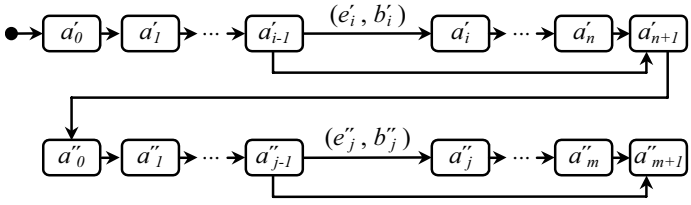


Abb. 7.5 Kripke-Struktur zweier sequenzieller Aktionenfolgen

7.7.6 Folgerung:

In TS3 kann $A' = a'_1, \ldots, a'_n$ mit der Kripke-Struktur $KS_{A'} = (S', S'_0, R', L')$ keine do-actions sein. Diese Folgerung ergibt sich aus der Konstruktion der Menge der Kripke-Zustände von $KS_{A'}$ in TS2 und aus der Konstruktion von R in TS3, denn aus $a'_{n+1} \notin S'$ folgt, dass $\{(a'_{n+1}, a''_0)\}$ in R undefiniert ist.

7.7.7 Definition und Bezeichnung (sequenzielle Verknüpfungen)

Die Verknüpfung zweier Kripke-Strukturen KS_1 und KS_2 , die sich nach TS3 aus zwei Aktionenfolgen A_1 und A_2 ergibt, wird hier *sequenzielle Verknüpfung* genannt und mit $KS_1 \oplus KS_2$ bezeichnet.

Die sequenzielle Verknüpfung der Kripke-Strukturen wird bei der Transformation der Zustandsdiagramme der UML verwendet. Dabei sind die Eigenschaften dieser Verknüpfung wichtig, die im Folgenden erläutert werden.

7.7.7.1 Eigenschaften der sequenziellen Verknüpfung

1. Die sequenzielle Verknüpfung von Kripke-Strukturen ist assoziativ, das heißt, für KS_1 , KS_2 und KS_3 gilt $KS_1 \oplus (KS_2 \oplus KS_3) = (KS_1 \oplus KS_2) \oplus KS_3$.
2. Die sequenzielle Verknüpfung von Kripke-Strukturen ist nicht kommutativ, das heißt, für KS_1 und KS_2 gilt nicht $KS_1 \oplus KS_2 = KS_2 \oplus KS_1$.

Zu 1. :

Es seien die Kripke-Strukturen $KS_1 = (S_1 , S_{0\,1} , R_1 , L_1)$, $KS_2 = (S_2 , S_{0\,2} , R_2 , L_2)$ und $KS_3 = (S_3 , S_{0\,3} , R_3 , L_3)$. Dann gelten:

1.1. $S_1 \cup (S_2 \cup S_3) = (S_1 \cup S_2) \cup S_3$,

1.2. Auf der linken Seite ist $S_{0\,2}$ in der Verknüpfung $(KS_2 \oplus KS_3)$ die Menge der Anfangszustände. Durch die Verknüpfung von KS_1 mit $(KS_2 \oplus KS_3)$ wird $S_{0\,1}$ zu der Menge der Anfangszustände der gesamten Verknüpfung. Auf der rechten Seite ist $S_{0\,1}$ die Menge der Anfangszustände von $(KS_1 \oplus KS_2)$.

Bei der Verknüpfung von $(KS_1 \oplus KS_2)$ mit KS_3 bleibt $S_{0\,1}$ als Menge der Anfangszustände der gesamten Verknüpfung.

Daraus folgt, dass in beiden Fällen $S_{0\,1}$ die Menge der Anfangszustände ist,

1.3. Es seien $S_1 = \{ a_{1_0} , a_{1_1} , \ldots , a_{1_{n+1}} \}$, $S_2 = \{ a_{2_0} , a_{2_1} , \ldots , a_{2_{m+1}} \}$ und $S_3 = \{ a_{3_0} , a_{3_1} , \ldots , a_{3_{p+1}} \}$. Auf der linken Seite der Gleichung ergibt sich durch die Verknüpfung $(KS_2 \oplus KS_3)$ die Transitionsrelation $R_2 \cup R_3 \cup \{(a_{2_{m+1}} , a_{3_0})\}$. Durch die Verknüpfung von KS_1 mit $(KS_2 \oplus KS_3)$ ergibt sich dann die Transitionsrelation $R_1 \cup R_2 \cup R_3 \cup \{(a_{2_{m+1}} , a_{3_0})\} \cup \{(a_{1_{n+1}} , a_{2_0})\}$ für die gesamte Verknüpfung.

Auf der rechten Seite ergibt sich durch die Verknüpfung $(KS_1 \oplus KS_2)$ die Transitionsrelation $R_1 \cup R_2 \cup \{(a_{1_{n+1}} , a_{2_0})\}$. Durch die Verknüpfung von

($KS_1 \oplus KS_2$) mit KS_3 ergibt sich dann die folgende Transitionsrelation für die gesamte Verknüpfung:

$R_1 \cup R_2 \cup \{(a_{1_{n+1}}, a_{2_0})\} \cup R_3 \cup \{(a_{2_{m+1}}, a_{3_0})\}$.

Folglich sind die beiden Transitionsrelationen identisch.

1.4. $L_1 \cup (L_2 \cup L_3) = (L_1 \cup L_2) \cup L_3$.

Damit wurde gezeigt, dass die sequenzielle Verknüpfung von Kripke-Strukturen, die sich aus den Aktionenfolgen ergeben, assoziativ ist, das heißt, die Gruppierung der beteiligten Kripke-Strukturen hat keine Auswirkung auf das Ergebnis.

Zu 2. :

Es seien $KS_1 = (S_1, S_{0_1}, R_1, L_1)$ und $KS_2 = (S_2, S_{0_2}, R_2, L_2)$. Die Menge der Anfangszustände ist für $KS_1 \oplus KS_2$ gleich S_{0_1} und für $KS_2 \oplus KS_1$ gleich S_{0_2}.

Daraus folgt, dass die sequenzielle Verknüpfung der Kripke-Strukturen, die sich aus den Aktionenfolgen ergeben, nicht kommutativ ist, das heißt, die Reihenfolge der gegebenen Kripke-Strukturen darf nicht vertauscht werden.

7.7.8 Transformation von zeitlich überlappenden Aktionenfolgen

In einem Zustand können zwei oder mehrere Aktionenfolgen vorkommen, die von dem Objekt zeitlich überlappend durchgeführt werden. Die Abbildung einer parallelen Ausführung mehrerer Aktionenfolgen wird in der Literatur unterschiedlich interpretiert und bezeichnet. Wird in jedem Schritt lediglich bei einer der parallelen Aktionenfolgen eine Aktion ausgeführt, so wird der Vorgang mit „Interleaving“ bezeichnet. Formale Beschreibungen derartiger Vorgänge sind z. B. in [Pin02, deR+01] gegeben. Finden in jedem Schritt alle ausführbaren Aktionen statt, so wird der Vorgang „multi-Step“ genannt, der unterschiedlich z. B. in [Pin02, dTd+96] formal beschrieben worden sind.

In einem parallelen Ablauf von Aktionenfolgen in einem Zustandsdiagramm der UML kann die Ausführung einer Aktion in einer Aktionenfolge mehr Zeit in Anspruch nehmen als die Ausführung einer anderen Aktion in der mitlaufenden Aktionenfolge, bzw. während sich das Objekt mit der Ausführung einer Aktion beschäftigt oder auf ein Ereignis für die Ausführung der darauf folgenden Aktion wartet, können in anderen Aktionenfolgen verschiedene Aktionen ausgeführt werden. Daher findet der Ablauf in allen Aktionenfolgen nicht in gleichen Schritten statt.

Für das Model-Checking ist es wichtig, alle möglichen Abläufe aufzustellen, die durch verschiedene Ausführungen entstehen können. Daher müssen die Aktionen einer Aktionenfolge mit Aktionen anderer paralleler Aktionenfolgen kombiniert werden, um die gleichzeitige Ausführung der beteiligten Aktionen abzubilden.

Die Kombination von Aktionen verschiedener Aktionenfolgen besteht darin, dass im ersten Schritt zu jeder Aktionenfolge die entsprechende Kripke-Struktur aufgestellt wird. Die Elemente der Zustandsmenge dieser Kripke-Strukturen sind die Aktionen der jeweiligen Aktionenfolgen. In dem nächsten Schritt wird unter den Zustandsmengen der entstandenen Kripke-Strukturen das kartesische Produkt gebildet, das die Menge aller möglichen Kombinationen der Aktionenfolgen ist. Das Ergebnis ist dann die Zustandsmenge der Kripke-Struktur des parallelen Ablaufs. In dieser Zustandsmenge ist der Anfangszustand der Kripke-Zustand, der sich aus der Kombination der Anfangszustände der einzelnen Kripke-Strukturen bildet.

Die Transitionen unter den kombinierten Zuständen entstehen dadurch, dass für jeden kombinierten Kripke-Zustand ein Folgezustand definiert wird, in dem es eine Folgeaktion für mindestens eine der Aktionen im kombinierten Kripke-Zustand gibt. Dadurch werden alle Fälle abgebildet, in denen in einer Aktionenfolge eine Aktion beendet und die Ausführung der Folgeaktion begonnen wird, während in den anderen Aktionenfolgen die Ausführung der bereits begonnenen Aktionen noch nicht beendet worden ist.

Die Beschriftungsfunktion der entstandenen Kripke-Struktur ist dann die Funktion, die jedem kombinierten Kripke-Zustand die Werte der entsprechenden Attribute zuordnet.

Formal lässt sich diese Vorgehensweise, wie folgt, definieren:

7.7.9 Transformationsschema 4 (TS4):

Es seien A' und A'' und ihre Kripke-Strukturen $KS_{A'}$ und $KS_{A''}$ wie in TS3 gegeben.

Für die Aktionenfolge $A'\,A''$, in der A' und A'' parallel ablaufen, ergibt sich die Kripke-Struktur $KS_{A'A''}$ durch (S, S_0, R, L) mit folgenden Eigenschaften:

1. $S = S' \times S''$,
2. $S_0 = S'_0 \times S''_0 = \{ a'_0\, a''_0 \}$,
3. $R = \{(a'_i\, a''_j , a'_k\, a''_r) \mid (a'_i , a'_k) \in R' \vee (a''_j , a''_r) \in R'' \}$ und
4. Es gilt $L = L' \times L''$ und $L : S \rightarrow \mathcal{P}(\mathcal{A}trr)$ liefert in jedem kombinierten Zustand die aktuellen Werte der Attribute in $\mathcal{A}trr$, die sich aus der Ausführung jeder Aktion ergeben.

Die Elemente in S sind kombinierte Kripke-Zustände aus Elementen von S' und S''. Da jedes Element in S lediglich die gleichzeitige Ausführung zweier Aktionen darstellt, bezeichnen die Zustandsnamen $a'_i\, a''_j$ und $a''_j\, a'_i$ das gleiche Verhalten, nämlich die gleichzeitige Ausführung von a'_i und a''_j. In der vorliegenden Arbeit werden $a'_i\, a''_j$ und $a''_j\, a'_i$ für gleich gehalten. Daher wird beispielsweise in Abschnitt 7.7.13 die Schreibweise $a'_i\, a''_j = a''_j\, a'_i$ verwendet. Im mathematischen Sinn ist die Beziehung zwischen $a'_i\, a''_j$ und $a''_j\, a'_i$ durch Isomorphie zu erklären, die durch $a'_i\, a''_j \cong a''_j\, a'_i$ bezeichnet wird. Bei

dieser Isomorphie kann für $a'_i\ a''_j$ und $a''_j\ a'_i$ gezeigt werden, dass es sich um eine echte Gleichheit handelt. Dieser Schritt kann aber nicht im Rahmen dieser Arbeit diskutiert werden.

Weiter sei erwähnt, dass $a'_{n+1}\ a''_{m+1} \in S$ der Kripke-Zustand ist, in dem die beiden Aktionenfolgen A' und A'' abgeschlossen sind. Dieser Kripke-Zustand ist für eine sequenzielle Verknüpfung mit anderen Kripke-Strukturen der Zustand, der mit dem Anfangszustand der nächsten Kripke-Struktur zu verbinden ist.

In den folgenden Beispielen wird die Kripke-Struktur, die sich nach dem genannten Schema aus zwei parallelen Aktionenfolgen ergibt, grafisch dargestellt:

7.7.10 Beispiel

Es sei z ein Zustand mit $act_1(z) = a_1\ ,\ a_2\ ,\ a_3$ und $act_2(z) = a_6\ ,\ a_7$ gegeben (Abb. 7.6). Aufgrund der Festlegung in Abschnitt 5.3.1.3 werden diese Aktionenfolgen parallel ausgeführt. Dabei enden $act_1(z)$ und $act_2(z)$ nach einmaliger Ausführung.

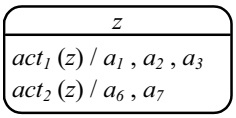


Abb. 7.6 Zustand mit parallelen Aktionenfolgen

Aus TS4 ergibt sich für den parallelen Ablauf von $act_1(z)$ und $act_2(z)$ eine Kripke-Struktur, deren Kripke-Zustände und Transitionen in Abb. 7.7 dargestellt ist. In dieser Abbildung gelten $a_4 = a_8 = a_\varepsilon$ und bezeichnen den Abschluss von $act_1(z)$ bzw. $act_2(z)$. Der kombinierte Zustand $a_4\ a_8$ zeigt, den Abschluss von beiden Aktionenfolgen.

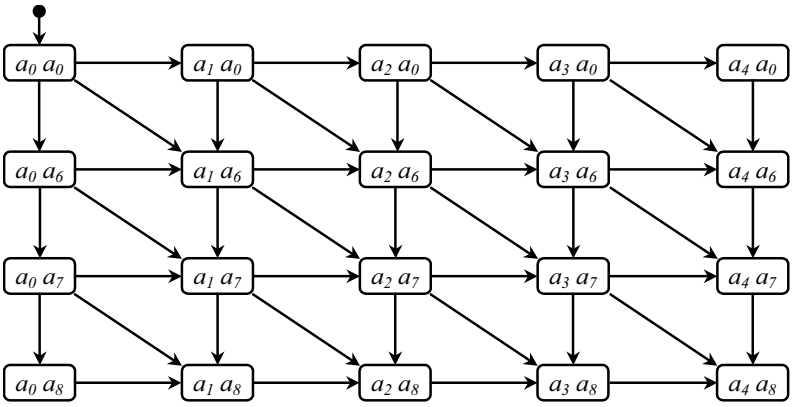


Abb. 7.7 mögliche Ausführungen der Aktionen von parallelen Aktionenfolgen

Existiert unter den parallelen Aktionenfolgen eines Zustands *z* die Aktionenfolge *do*(*z*), dann werden die Zustandsmenge des parallelen Ablaufs und die Transitionsrelation in der entsprechenden Kripke-Struktur wie im folgenden Beispiel gebildet.

7.7.11 Beispiel

Es sei *z* ein Zustand mit $do(z) = a_1 , a_2 , a_3$ und $act(z) = a_6 , a_7$ gegeben (Abb. 7.8). Aufgrund der Festlegung in Abschnitt 5.3.1.4 werden diese Aktionenfolgen zeitlich überlappend ausgeführt. Dabei endet $act(z) = a_6 , a_7$ nach einmaliger Ausführung, während sich $do(z) = a_1 , a_2 , a_3$ wiederholt, solange das Objekt sich in *z* befindet.

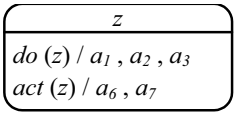


Abb. 7.8 Zustand mit parallelen do(z) und act(z)

Für den parallelen Ablauf von *do*(*z*) und *act*(*z*) ergibt sich aus TS2 und TS3 eine Kripke-Struktur, deren Kripke-Zustände und Transitionen in Abb. 7.9 dargestellt wurde. In dieser Abbildung gilt $a_8 = a_\varepsilon$ und bezeichnet den Abschluss von *act* (*z*).

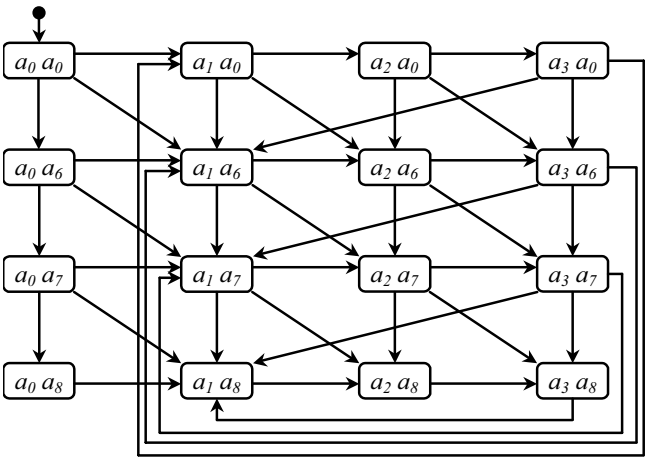


Abb. 7.9 mögliche Ausführungen der Aktionen von parallelen do(z) und act(z)

7.7.12 Definition und Bezeichnung (parallele Verknüpfung)

Die Verknüpfung zweier Kripke-Strukturen KS_1 und KS_2 , die sich nach TS4 aus zwei Aktionenfolgen A_1 und A_2 ergeben, wird hier *parallele Verknüpfung* genannt und mit $KS_1 \otimes KS_2$ bezeichnet.

Bei der Transformation der Zustandsdiagramme der UML kann die parallele Verknüpfung der Kripke-Strukturen verwendet werden. Daher werden im Folgenden die Eigenschaften dieser Verknüpfung, die für die Transformation der Zustandsdiagramme der UML relevant sind, erläutert.

7.7.13 Eigenschaften der parallelen Verknüpfung

1. Die parallele Verknüpfung von Kripke-Strukturen ist assoziativ, das heißt, für KS_1 , KS_2 und KS_3 gilt $KS_1 \otimes (KS_2 \otimes KS_3) = (KS_1 \otimes KS_2) \otimes KS_3$.
2. Die parallele Verknüpfung von Kripke-Strukturen ist kommutativ, das heißt, für KS_1 und KS_2 gilt $KS_1 \otimes KS_2 = KS_2 \otimes KS_1$.

Zu 1. :

Es seien $KS_1 = (S_1 , S_{0\,1} , R_1 , L_1)$, $KS_2 = (S_2 , S_{0\,2} , R_2 , L_2)$ und $KS_3 = (S_3 , S_{0\,3} , R_3 , L_3)$ die Kripke-Strukturen zu Aktionenfolgen A_1 , A_2 und A_3 mit $A_1 = a_{1_1} , \ldots , a_{1_n}$, $A_2 = a_{2_1} , \ldots , a_{2_m}$ und $A_3 = a_{3_1} , \ldots , a_{3_p}$. Dann gelten:

1.1. $S_1 \times (S_2 \times S_3) = (S_1 \times S_2) \times S_3$,

1.2. Für die Menge der Anfangszustände, die sich von der linken und von der rechten Seite ergeben, gilt $S_{0\,1} \times (S_{0\,2} \times S_{0\,3}) = (S_{0\,1} \times S_{0\,2}) \times S_{0\,3}$.

1.3. Bei der Transitionsrelation gilt für die beiden Seiten:

$R = \{(a_{1_i} a_{2_j} a_{3_q} , a_{1_k} a_{2_r} a_{3_l}) \mid (a_{1_i} , a_{1_k}) \in R_1 \vee (a_{2_j} , a_{2_r}) \in R_2 \vee (a_{3_q} , a_{3_l}) \in R_3 \}$ und

1.4. $L_1 \times (L_2 \times L_3) = (L_1 \times L_2) \times L_3$.

Damit wurde gezeigt, dass wie die sequenzielle Verknüpfung auch die parallele Verknüpfung von Kripke-Strukturen, die sich aus den Aktionenfolgen ergeben, assoziativ ist, das heißt, die Gruppierung der beteiligten Kripke-Strukturen hat keine Auswirkung auf das Ergebnis.

Zu 2. :

Es seien $KS_1 = (S_1 , S_{0\,1} , R_1 , L_1)$ und $KS_2 = (S_2 , S_{0\,2} , R_2 , L_2)$ die Kripke-Strukturen zu Aktionenfolgen $A_1 = a_{1_1} , \ldots , a_{1_n}$ und $A_2 = a_{2_1} , \ldots , a_{2_m}$, dann gelten:

2.1. $S_1 \times S_2 = \{ a_{1_i} a_{2_j} \mid a_{1_i} \in S_1 \wedge a_{2_j} \in S_2 \} = \{ a_{2_j} a_{1_i} \mid a_{2_j} \in S_2 \wedge a_{1_i} \in S_1 \} = S_2 \times S_1$.

Die Korrektheit der zweiten Gleichung folgt aus $a_{1_i} a_{2_j} = a_{2_j} a_{1_i}$, denn $a_{1_i} a_{2_j}$ und $a_{2_j} a_{1_i}$ stellen, wie in Abschnitt 7.7.9 erläutert wurde, den parallelen Ablauf von a_{1_i} und a_{2_j} dar.

2.2. Ebenfalls gilt $S_{0\,1} \times S_{0\,2} = \{ a_{1_0}\, a_{2_0} \} = \{ a_{2_0}\, a_{1_0} \} = S_{0\,2} \times S_{0\,1}$. Hier sind a_{1_0} und a_{2_0} die Anfangszustände, die durch das Transformationsschema TS1 bei A_1 und A_2 hinzugefügt worden sind.

2.3. Bei der Transitionsrelation gilt

$$\{(a_{1_i}\, a_{2_j}, a_{1_k}\, a_{2_r}) \mid (a_{1_i}, a_{1_k}) \in R_1 \vee (a_{2_j}, a_{2_r}) \in R_2 \} =$$

$$\{(a_{2_j}\, a_{1_i}, a_{2_r}\, a_{1_k}) \mid (a_{2_j}, a_{2_r}) \in R_2 \vee (a_{1_i}, a_{1_k}) \in R_1 \}$$ und

2.4. $L_1 \times L_2 = L_2 \times L_1$.

Damit wurde gezeigt, dass die parallele Verknüpfung von Kripke-Strukturen kommutativ ist. Daraus folgt, dass bei der parallelen Verknüpfung der Kripke-Strukturen, die sich aus den Aktionenfolgen ergeben, eine Änderung der Reihenfolge der gegebenen Kripke-Strukturen das Ergebnis nicht ändert.

Mit Hilfe von sequenzieller und paralleler Verknüpfung von Kripke-Strukturen können aus den Kripke-Strukturen einzelner Aktionenfolgen Kripke-Strukturen für Zustände und Transitionen bzw. für das gesamte Zustandsdiagramm konstruiert werden. Hier wird zuerst die Transformation von Transitionen erläutert.

7.7.14 Transformation von Transitionen

Eine Transition veranlasst das Verlassen eines Zustands und den Eintritt in den Folgezustand. Die Transition selbst wird von der Austrittsbedingung des aktiven Zustands gesteuert. Wenn der Zustand aktiv und seine Austrittsbedingung erfüllt ist, findet die Transition statt. In Kapitel fünf wurde eine Transition in Form von $t = z_i \xrightarrow{e[b]/A} z_j$ definiert, die den Zustandswechsel von z_i mit der Austrittsbedingung (e, b) nach z_j bewirkt. Dabei finden nicht nur das Verlassen von z_i und der Eintritt in z_j statt, sondern alle Oberzustände von z_i werden verlassen, die keine Oberzustände von z_j sind, und alle Oberzustände von z_j werden aktiviert, die keine Oberzustände von z_i sind. Gemäß Abschnitt 5.4.2 erfolgen bei diesem Vorgang die folgenden Ausführungen in der gegebenen Reihenfolge:

1. Ausführung der exit-actions der zu verlassenden Oberzustände von z_i,
2. Ausführung der mit der Transition verbundenen Aktionenfolge A und
3. Ausführung der entry-actions aller einzutretenden Oberzustände von z_j.

In dem nächsten Abschnitt werden alle möglichen Anordnungen von den Aktionenfolgen eines Zustands und die Beziehungen zwischen den entry-actions bzw. exit-actions eines Zustands und den entry-actions bzw. exit-actions seiner Ober- bzw. Teilzustände betrachtet. Dabei wird auch die Abbildung von möglichen Ausführungen von entry-actions und exit-actions, die infolge von Transitionen entstehen, behandelt. Damit wer-

den die Ausführung der exit-actions der zu verlassenden Oberzustände von z_i und die Ausführung der entry-actions aller einzutretenden Oberzustände von z_j in den beteiligten Zuständen abgebildet.

Daher wird die Kripke-Struktur KS_t für die Transition $t = z_i \xrightarrow{e[b]/A} z_j$ nur aus der sequenziellen Verknüpfung der Kripke-Strukturen $KS_{ex(z_I)}$, KS_A und $KS_{en(z_J)}$ gebildet, wobei $KS_{ex(z_I)}$ die Kripke-Struktur zu exit-actions des letzten zu verlassenden Oberzustands von z_i , KS_A die Kripke-Struktur zur Aktionenfolge A und $KS_{en(z_J)}$ die Kripke-Struktur zu entry-actions des ersten einzutretenden Oberzustands von z_j sind, das bedeutet, es gilt $KS_t = KS_{ex(z_I)} \oplus KS_A \oplus KS_{en(z_J)}$.

Da die sequenzielle Verknüpfung nicht kommutativ ist, darf für KS_t die Reihenfolge der zu verknüpfenden Kripke-Strukturen nicht vertauscht werden, jedoch wegen der Assoziativität ist gleichgültig, ob das Ergebnis von $KS_{ex(z_I)} \oplus KS_A$ mit $KS_{en(z_J)}$ oder $KS_{ex(z_I)}$ mit dem Ergebnis von $KS_A \oplus KS_{en(z_J)}$ verknüpft wird.

Unter Anwendung von sequenzieller bzw. paralleler Verknüpfung können für die einzelnen Zustände eines Zustandsdiagramms Kripke-Strukturen entworfen werden. Dafür muss zunächst der Zustand in seine Aktionenfolgen und Teilzustände zerlegt werden.

7.7.15 Zerlegung eines Zustands in seine Aktionenfolgen

In Kapitel fünf wurde gezeigt, dass in einem Zustand z Aktionenfolgen $en(z)$, $act_1(z)$ ... $act_k(z)$, $do(z)$ und $ex(z)$ vorkommen können und dass z Teilzustände besitzen kann, die in $include(z)$ definiert sind. Dabei werden die Aktivitäten $act_1(z)$ ... $act_k(z)$, $do(z)$ und die in $include(z)$ nach dem Beenden von $en(z)$ und vor dem Ausführen von $ex(z)$ zeitlich überlappend ausgeführt. Bezüglich $en(z)$ und $ex(z)$ wurde festgelegt, dass diese Aktionenfolgen ohne Unterbrechung ausgeführt werden, während $act_1(z)$ ... $act_k(z)$ und $do(z)$ unterbrochen werden, sobald die Austrittsbedingung (e, b) des Zustands z erfüllt ist.

Die genannten Eigenschaften dieser Aktivitäten führen zu einer ersten Unterteilung des Zustands z . Diese Unterteilung besteht aus den Teilzuständen z_{entry} mit der Aktionenfolge $en(z)$, $z_{actions}$ mit den Aktionenfolgen $act_1(z)$ ... $act_k(z)$, $do(z)$ und den Aktivitäten in $include(z)$ und z_{exit} mit der Aktionenfolge $ex(z)$. Da $en(z)$ und $ex(z)$ ohne Unterbrechung ausgeführt werden, reagiert das Objekt nur dann auf (e, b), wenn sich das Objekt in dem Teilzustand $z_{actions}$ befindet. Abb. 7.10 veranschaulicht diese Unterteilung.

Die Pfeile in dieser Abb. zeigen die sequenzielle Verknüpfung zwischen Kripke-Strukturen, die aus z_{entry} , $z_{actions}$ und z_{exit} entstehen.

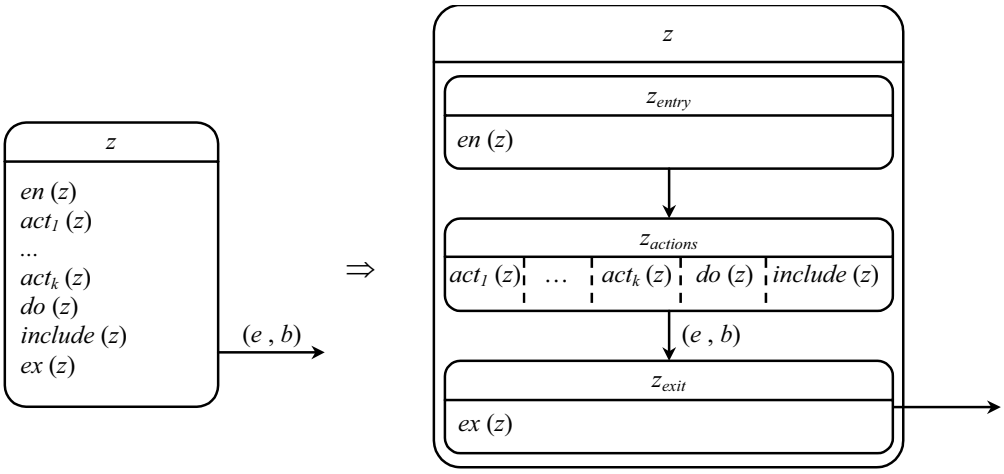


Abb. 7.10 Zustand in einem UML-Zustandsdiagramm und seine Unterteilung

7.7.16 Zerlegung von Elementen in *include*(*z*)

Die Elemente in *include*(*z*) sind entweder parallele oder nichtparallele Teilzustände von z . Nach der Zerlegung der einzelnen Teilzustände entstehen Verknüpfungen, die sich aus den definierten Transitionen unter den Teilzuständen ergeben. In den folgenden Abschnitten werden die Vorgehensweisen bei der Zerlegung der nichtparallelen und der parallelen Teilzustände erläutert.

7.7.17 Zerlegung von nichtparallelen Teilzuständen

Sind z_1 , ... , z_n nichtparallele Teilzustände von z mit der Austrittsbedingung (e, b), so werden zuerst die einzelnen Teilzustände in *include*(*z*) zerlegt. Danach werden $z_{i\ exit}$, A und $z_{j\ entry}$ für jede Transition $z_i \xrightarrow{e[b]/A} z_j$ mit $1 \le i, j \le n$ miteinander verknüpft. Bei der Initialtransition $\xrightarrow{e[b]/A} z_j$ werden z_{entry} , A und $z_{j\ entry}$ verknüpft. Weiter werden bei der Finaltransition $z_i \xrightarrow{e[b]/A}$ die Teilzustände $z_{i\ exit}$, A und z_{exit} miteinander verknüpft.

Dieses Verfahren wird auch bei den Zuständen ohne einen Oberzustand angewendet.

Außer den genannten Verknüpfungen entstehen bei den Teilzuständen eines Zustands weitere Verknüpfungen durch die Austrittsbedingungen der Oberzustände bzw. durch die direkten Eintritte in die Teilzustände. In Kapitel fünf wurde erläutert, dass die Austrittsbedingungen eines Zustands auch für seine Teilzustände gelten, und dass jeder Eintritt in einen Teilzustand den Eintritt in alle seine nicht aktiven Oberzustände bewirkt. Dementsprechend wurde gezeigt, dass ein Austritt aus einem Zustand mit der Ausführung einer Reihe von exit-actions verbunden ist, die den Teilzuständen des Zustands gehören. Durch derartige Ausführungen entstehen bei Zerlegung eines Zustands Verknüpfungen zwischen den exit-actions der Teilzustände und den exit-actions des gege-

benen Zustands. Das heißt, bei $(e, b) \neq (e_\varepsilon , b_\varepsilon)$ wird unabhängig davon, ob eine Finaltransition definiert ist, zusätzlich zu den Verknüpfungen, die durch die Finaltransition zustande kommen, für alle z_i mit $1 \leq i \leq n$ noch $z_{i\ exit}$ mit z_{exit} verknüpft, um die sequenzielle Ausführung der Aktionenfolgen $ex(z_i)$ und $ex(z)$ abzubilden.

Das entsprechende gilt auch für die entry-actions eines Zustands, denn bei einem Eintritt in einen Teilzustand mit nicht aktiven Oberzuständen erfolgt eine sequenzielle Ausführung von entry-actions statt, die den nicht aktiven Oberzuständen gehören. Daher entstehen Verknüpfungen zwischen den entry-actions des Oberzustands und den entry-actions des zu aktivierenden Teilzustands. Ist der Zustand ein History-State, so werden die entry-actions des Oberzustands mit den entry-actions aller Teilzustände verknüpft.

In Abb. 7.11 ist ein Zustand z mit zwei Teilzuständen z_1 und z_2 gegeben. Der Austritt aus z erfolgt durch die Austrittsbedingung (e, b). Dabei wird z verlassen, sobald (e, b) erfüllt ist. In diesem Fall kann sich das Objekt in z_1 bzw. in z_2 befinden. Daraus folgt eine Verknüpfung der exit-actions von z_1 bzw. z_2 mit den exit-actions von z .

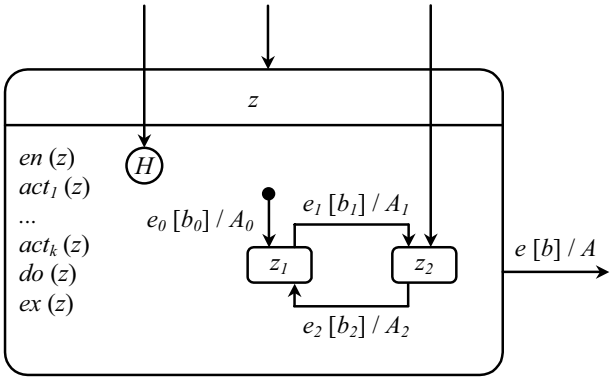


Abb. 7.11 Zustand mit einer Austrittsbedingung und drei Eintrittsmöglichkeiten

Weiter ist der Eintritt in z über drei Wege möglich. Der Eintritt über den Anfangszustand z_1 ist der reguläre Fall. Ein weiterer Eintritt ist über den Teilzustand z_2 gegeben und schließlich ist ein Eintritt über den History-State H möglich.

In Abb. 7.12, die die Zerlegung von z darstellt, entsprechen die Verknüpfungen zwischen $z_{1\ exit}$ bzw. $z_{2\ exit}$ und z_{exit} dem Austritt aus z_1 bzw. z_2 , wenn (e, b) erfüllt ist und z verlassen wird. Aufgrund dieser Verknüpfungen zwischen exit-actions von z_1 bzw. z_2 mit den exit-actions von z genügt es, wie bei der Transformation einer Transition erläutert wurde, dass für eine aus z bzw. aus einem seiner Teilzustände ausgehende Transition nur die exit-actions des letzten zu verlassenden Oberzustands mit der Aktionenfolge der Transition verknüpft werden.

Die Verknüpfung zwischen z_{entry} und $z_{1\ entry}$ über A_0 entsteht durch den regulären Eintritt in z. Die direkten Verknüpfungen zwischen z_{entry} und $z_{1\ entry}$ bzw. $z_{2\ entry}$ kommen durch den Eintritt über den History-State zustande. Dabei vertritt die Verknüpfung zwischen

z_{entry} und $z_{2\ entry}$ zugleich die dritte Eintrittsmöglichkeit über z_2 . Wegen dieser Verknüpfungen zwischen z_{entry} und $z_{1\ entry}$ bzw. $z_{2\ entry}$ genügt es dann für jede in z bzw. in einem seiner Teilzustände endende Transition, dass die Aktionenfolge der Transition nur mit den entry-actions des ersten einzutretenden Oberzustands verknüpft wird.

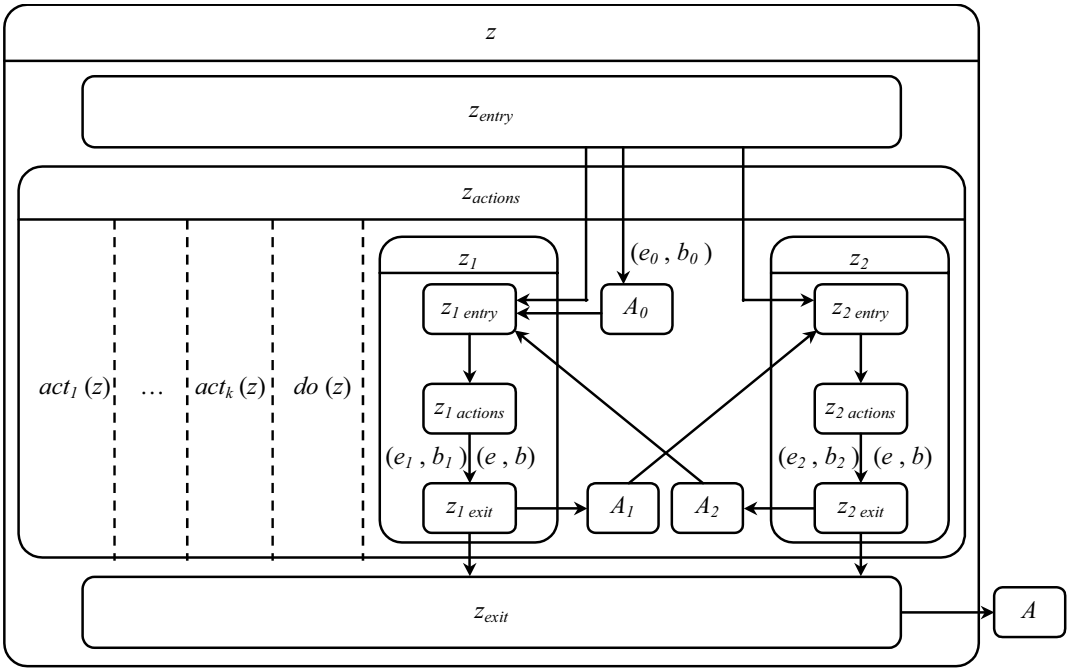


Abb. 7.12 Zerlegung des Zustands aus der Abb. 7.11

In Abb. 7.12 entsprechen die Verknüpfungen zwischen $z_{1\ exit}$, A_1 und $z_{2\ entry}$ der Transition zwischen den Zuständen z_1 und z_2 aus der Abb. 7.11. Ebenso verkörpern die Verknüpfungen zwischen $z_{1\ exit}$ bzw. $z_{2\ exit}$, z_{exit} und A die Transition, über die der Zustand z verlassen wird.

Anhand dieser Zerlegung kann aus den Kripke-Strukturen der Aktionenfolgen der Teilzustände A_0 , $z_{1\ entry}$, $z_{1\ actions}$ etc. eine Kripke-Struktur für alle Elemente in *include*(z) konstruiert werden. Diese Kripke-Struktur bildet sich aus den sequenziellen Verknüpfungen von Kripke-Strukturen einzelner Aktionenfolgen in den Teilzuständen. Dabei ist die Kripke-Struktur der Aktionenfolge der Defaulttransition A_0 die erste Kripke-Struktur. Damit ist die Menge der Anfangszustände dieser Kripke-Struktur die Menge der Anfangszustände der Kripke-Struktur für alle Elemente in *include*(z).

Die Pfeile bestimmen dann die nächste zu verknüpfende Kripke-Struktur. Mit den sequenziellen Verknüpfungen von Kripke-Strukturen der Aktionenfolgen der Teilzustände wird solange fortgesetzt, bis Kripke-Strukturen aller Aktionenfolgen verknüpft und alle Pfeile außer Verbindungspfeilen zwischen entry- bzw. exit-actions des Oberzustands und den Aktionenfolgen der Teilzustände in *include*(z) berücksichtigt worden sind.

Für eine vollständige Zerlegung von z werden in einem weiteren Schritt die Teilzustände in $z_{actions}$, die jetzt Teilzustände von parallelen Regionen sind, miteinander kombiniert. Das Kombinieren von Teilzuständen paralleler Regionen wird in dem nächsten Abschnitt erläutert.

7.7.18 Zerlegung von parallelen Regionen

Sind z_1 , ... , z_n parallele Teilzustände von z , so werden zuerst die einzelnen Teilzustände in *include*(z), die selbst aus nichtparallelen Teilzuständen bestehen, nach dem oben beschriebenen Verfahren zerlegt. Da die Aktivitäten in den parallelen Regionen zeitlich überlappend stattfinden, werden die Teilzustände jeder parallelen Region, die die internen Aktivitäten eines Teilzustands in dieser Region bzw. die Ausführung der Aktionenfolge einer Transition repräsentieren, mit den Teilzuständen der benachbarten Regionen kombiniert. Zu diesem Zweck wird unter den Mengen der Teilzustände der benachbarten parallelen Regionen das kartesische Produkt gebildet. Dadurch werden alle möglichen Kombinationen von Ausführungen der Aktivitäten in den parallelen Regionen nachgebildet. Ein kombinierter Zustand, der sich auf dieser Weise aus zwei oder mehreren Teilzuständen verschiedener Regionen zusammensetzt, steht für eine gleichzeitige Ausführung von Aktivitäten der beteiligten Teilzustände. Daher gelten die ursprünglichen Verknüpfungen zwischen jedem einzelnen Teilzustand mit anderen Teilzuständen zugleich für den kombinierten Zustand. Folglich werden in einem weiteren Schritt die kombinierten Zustände gemäß den Transitionen verknüpft, die in jeder Region für die Zustände definiert sind.

Die Vorgehensweise zum Kombinieren der Zustände der parallelen Regionen und zum Übertragen der Verknüpfungen zwischen den einzelnen Zuständen auf die kombinierten Zustände wird in dem folgenden Verfahren erläutert.

7.7.19 Kombinieren von Teilzuständen paralleler Regionen

In diesem Abschnitt wird anhand eines Zustands mit parallelen Regionen, die selbst weitere nichtparallele Teilzustände haben, ein Verfahren zur Kombination von Teilzuständen paralleler Regionen vorgestellt. Dabei wird angenommen, dass die nichtparallelen Teilzustände in jeder Region die in Abschnitt 7.7.17 genannte Zerlegung bereits erfahren haben.

Es seien z ein Zustand und z' und z'' seine parallelen Regionen. Weiter seien z'_1 , … , z'_n nichtparallele Teilzustände von z' und z''_1 , … , z''_m nichtparallele Teilzustände von z''. Die Kombination von z'_1 , … , z'_n mit z''_1 , … , z''_m ergibt sich in den folgenden Schritten:

1. Es wird eine Tabelle mit n + 1 Zeilen und m + 1 Spalten gebildet. Die Zeilen werden mit 0, ... , n und die Spalten mit 0, ... , m gekennzeichnet.

2. z'_1 , … , z'_n werden in die erste Spalte und z''_1 , … , z''_m werden in die erste Zeile der Tabelle eingetragen. Das erste Feld in der ersten Zeile und in der ersten Spalte erhält keine Eintragung.
3. In jedem Feld der Tabelle wird ein kombinierter Zustandsname geschrieben, der sich aus den Zustandsnamen der entsprechenden Zeile und Spalte ergibt. Damit steht $z'_i z''_j$ in dem Feld der *i*-ten Zeile und der *j*-ten Spalte
4. Alle Transitionspfeile zwischen z'_1 , … , z'_n werden entsprechend ihrer Definition auf die erste Spalte und alle Transitionspfeile zwischen z''_1 , … , z''_m werden entsprechend ihrer Definition auf die erste Zeile der Tabelle übertragen.
5. Aus der ersten Zeile wird jeder Pfeil zwischen der *i*-ten und der *j*-ten Spalte in alle Zeilen zwischen die *i*-te und die *j*-te Spalte übertragen.
6. Aus der ersten Spalte wird jeder Pfeil zwischen der *i*-ten und der *j*-ten Zeile in alle Spalten zwischen die *i*-te und die *j*-te Zeile übertragen.
7. In jedem Feld mit der Eintragung $z'_i z''_j$ werden alle ausgehenden und nicht parallelen Pfeile paarweise ausgewählt. Zeigt ein ausgewähltes Pfeilpaar auf die Zeile mit der Nummer *p* und auf die Spalte mit der Nummer *q*, so wird ein neuer Pfeil von $z'_i z''_j$ nach $z'_p z''_q$ gezeichnet.
8. Die Pfeile in der Tabelle mit Ausnahme der Pfeile aus der ersten Zeile und der ersten Spalte sind die Transitionspfeile bzw. Verknüpfungen zwischen den kombinierten Zuständen.

Für die Transitionen, die sich aus den Austrittsbedingungen des Oberzustands ergeben, werden die kombinierten Zustände, in denen ausschließlich die exit-actions der Teilzustände vorkommen, mit den exit-actions des Oberzustands verknüpft, denn beim Verlassen des Oberzustands werden alle parallelen Regionen gleichzeitig verlassen und dabei werden in jeder Region die exit-actions des aktiven Teilzustands ausgeführt.

Im Gegensatz zu Austritten werden für jeden Eintritt in den Oberzustand die entry-actions des Oberzustands mit jedem kombinierten Zustand verknüpft, in dem mindestens eine Aktionenfolge einer Defaulttransition vorkommt, denn nach dem Ausführen der entry-actions des Oberzustands können die Anfangszustände der parallelen Regionen zeitlich überlappend aktiviert werden, das heißt, die Anfangszustände werden aktiviert, bei denen die Ausführungsbedingungen der Defaulttransitionen erfüllt sind.

Wie in Kapitel fünf erläutert wurde, können die Zustände mit parallelen Regionen auch über ihre Teilzustände aktiviert werden. Dabei werden gleichzeitig alle parallelen Regionen des Oberzustands aktiviert, wenn die Transition in einen Teilzustand stattfindet. Für derartige Transitionen werden die entry-actions des Oberzustands mit den kombinierten Zuständen verbunden, in denen die entry-actions des Teilzustands vorkommen, in dem die Transition endet.

Nach dem Kombinieren von Teilzuständen der parallelen Regionen kann aus den Kripke-Strukturen einzelner Aktionenfolgen der Teilzustände eine Kripke-Struktur für alle parallelen Regionen konstruiert werden. Dabei wird zuerst für jede einzelne Aktionenfolge in jedem Teilzustand eine Kripke-Struktur entworfen. Danach wird für jeden kombinierten Zustand eine Kripke-Struktur aus den parallelen Verknüpfungen von Kripke-Strukturen der beteiligten Aktionenfolgen gebildet. Die entstandenen Kripke-Strukturen werden dann sequenziell verknüpft. Bei der Kripke-Struktur für alle parallelen Regionen ist die Kripke-Struktur der parallelen Aktionenfolgen der Defaulttransitionen die erste zu verknüpfende Kripke-Struktur und die Menge der Anfangszustände dieser Kripke-Struktur ist die Menge der Anfangszustände der Kripke-Struktur für alle Elemente in *include*(z).

Die weiteren sequenziellen Verknüpfungen ergeben sich aus den Pfeilen unter den kombinierten Zuständen. Die sequenziellen Verknüpfungen werden solange fortgesetzt, bis Kripke-Strukturen aller kombinierten Zustände verknüpft und alle Pfeile außer Verbindungspfeilen zwischen entry- bzw. exit-actions des Oberzustands und den kombinierten Zuständen berücksichtigt worden sind.

Das Verfahren zum Kombinieren der Teilzustände paralleler Regionen wird in dem folgenden Beispiel anhand eines Zustands mit zwei parallelen Regionen, die jeweils nichtparallele Teilzustände haben, veranschaulicht.

7.7.20 Beispiel

Es seien z ein Zustand und z_6 und z_7 mit $z_6 \parallel z_7$ seine Teilzustände, wie in Abb. 7.13 dargestellt wird. Weiter seien z_1 , z_2 und z_3 nichtparallele Teilzustände von z_6 und z_4 , z_5 und *End* (Finalzustand) nichtparallele Teilzustände von z_7 . Die Transitionen zwischen den Teilzuständen sind ebenso in Abb. 7.13 gegeben.

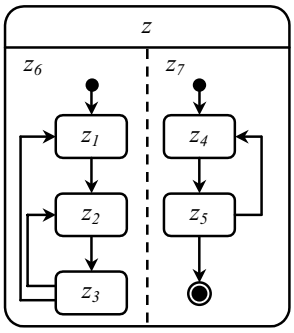
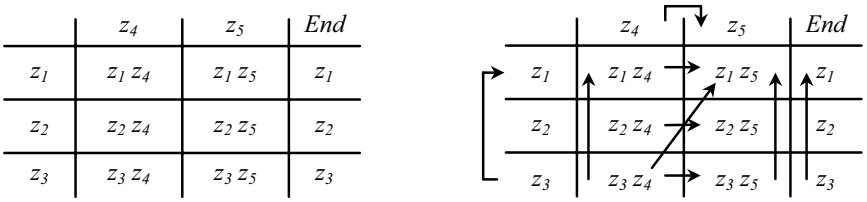


Abb. 7.13 Zustand mit parallelen Regionen

Für das Kombinieren von Zuständen z_1 , z_2 und z_3 mit z_4 , z_5 und *End* wird zuerst eine Tabelle mit vier Zeilen und vier Spalten erstellt. In dieser Tabelle werden gemäß den Schritten 2 und 3 die Zustandsnamen z_1 , z_2 und z_3 in die erste Spalte und die Zustands-

namen z_4 , z_5 und *End* in die erste Zeile eingetragen. Die Eintragungen in den weiteren Feldern dieser Tabelle ergeben sich aus den Kombinationen der genannten Zustandsnamen, die sich aus den Zeilen- bzw. Spaltennamen bilden (Abb. 7.14 a)).

In einem weiteren Schritt werden die Transitionen zwischen den einzelnen Zuständen gemäß dem Schritt 4 in die erste Zeile bzw. in die erste Spalte übertragen. Damit die Darstellung in Abb. 7.14 b) übersichtlich bleibt, werden dort nicht alle Transitionspfeile, sondern nur die von z_3 nach z_1 und die von z_4 nach z_5 übertragen. Gemäß Schritten fünf und sechs werden der Transitionspfeil von z_3 nach z_1 horizontal auf alle Spalten und der Transitionspfeil von z_4 nach z_5 vertikal auf alle Zeilen übertragen.



a) Kartesisches Produkt der Zustände b) Übertragung der Transitionspfeile

Abb. 7.14 Kombinieren von Zuständen paralleler Regionen

Gemäß dem Schritt sieben werden in jedem Feld dieser Tabelle alle ausgehenden und nicht parallelen Pfeile paarweise ausgewählt. Abb. 7.14 b) zeigt beispielhaft in dem Feld mit der Eintragung $z_3\ z_4$ die Pfeile von $z_3\ z_4$ nach $z_3\ z_5$ und von $z_3\ z_4$ nach $z_1\ z_4$, die zwei ausgehende und nicht parallele Pfeile sind. Da diese Pfeile auf die Zeile mit der Eintragung z_1 und auf die Spalte mit der Eintragung z_5 zeigen, so ergibt sich ein Pfeil von dem Feld mit der Eintragung $z_3\ z_4$ zum Feld mit der Eintragung $z_1\ z_5$. Die vollständige Übertragung aller Transitionspfeile ist in Abb. 7.15 dargestellt worden.

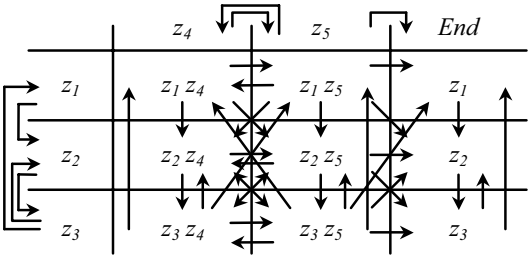


Abb. 7.15 Übertragung der Transitionen auf die kombinierten Zustände

Damit ergeben sich die Verknüpfungen zwischen den kombinierten Zuständen aus den Pfeilen zwischen den Feldern der Tabelle. Abb. 7.16 zeigt alle kombinierten Zustände und ihre Verknüpfungen für den Zustand *z* im Beispiel 7.7.20.

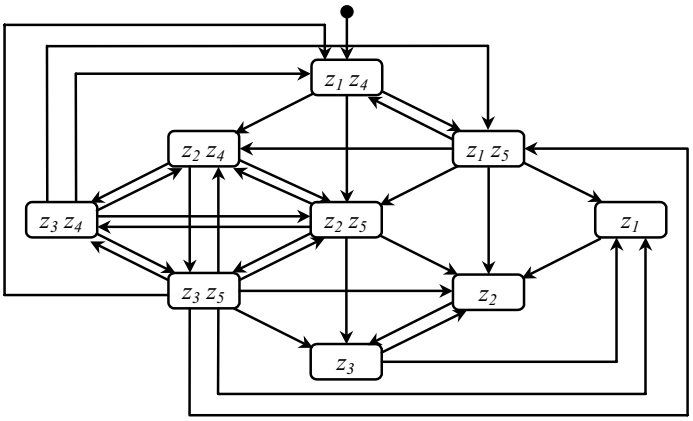


Abb. 7.16 Kombinierte Zustände und ihre Verknüpfungen

In diesem Beispiel sind kein Austritt aus dem Oberzustand, kein Eintritt in den Oberzustand bzw. keine Aktivierung des Oberzustands über einen Teilzustand gegeben. Daher werden keine weiteren Verknüpfungen zwischen den entry- bzw. exit-actions des Oberzustands und den entry- bzw. exit-actions der kombinierten Zuständen gezeigt.

Die oben beschriebenen Verfahren ermöglichen die Zerlegung der Teilzustände eines Zustands und legen die Verknüpfungen zwischen den Aktionenfolgen des Zustands und den Aktionenfolgen seiner Teilzustände fest. Mit Hilfe dieser Verfahren können alle Aktivitäten eines Zustands in einzelne Aktionenfolgen zerlegt werden und parallele, nicht parallele, vollständige oder nicht vollständige Abläufe dieser Aktivitäten nachgebildet werden. Dieser Schritt vereinfacht die Transformation des Zustands in eine geeignete Kripke-Struktur.

7.7.21 Transformation eines Zustands

Für die Transformation eines Zustands in eine Kripke-Struktur wird zuerst der Zustand in seine Aktionenfolgen zerlegt. In einem weiteren Schritt wird mit Hilfe von TS1 oder TS2 zu jeder nicht leeren Aktionenfolge eine Kripke-Struktur konstruiert. Danach werden die Kripke-Strukturen miteinander verknüpft.

Die Verknüpfungen zwischen den entstandenen Kripke-Strukturen, die sich aus den in Abschnitt 7.7.15 genannten Teilzuständen ergeben, hängen jedoch davon ab, ob der Zustand nach dem vollständigen Beenden seiner internen Aktivitäten oder aufgrund einer Austrittsbedingung und folglich mit Unterbrechung der internen Aktivitäten verlassen wird. In beiden Fällen bestimmt auch die Anordnung der internen Aktivitäten die Verknüpfung zwischen den entstandenen Kripke-Strukturen.

Um alle genannten Fälle zu berücksichtigen, werden in Transformationsschema fünf zur Transformation eines Zustands z mit der Austrittsbedingung (e, b) diese Fälle anhand

einer Fallunterscheidung erläutert. Dabei steht $(e, b) = (e_\varepsilon , b_\varepsilon)$ für den Fall, in dem z nach dem vollständigen Beenden seiner internen Aktivitäten verlassen wird, während $(e, b) \neq (e_\varepsilon , b_\varepsilon)$ den Fall zeigt, in dem z aufgrund seiner Austrittsbedingung verlassen werden kann, ohne seine internen Aktivitäten vollständig beenden zu müssen. Weiter werden die Fälle, in denen z keinen bzw. einige Teilzustände hat durch $include(z) = \varnothing$ bzw. $include(z) \neq \varnothing$ unterschieden.

7.7.22 Transformationsschema 5 (TS5):

Es sei z ein Zustand mit der Austrittsbedingung (e, b) und den Aktionenfolgen $en(z)$, $act_1(z)$ ... $act_k(z)$, $do(z)$ und $ex(z)$. Weiter sei $include(z)$ die Menge der Teilzustände in z.

Für z lässt sich die Kripke-Struktur $KS_{(z)} = (S_{(z)} , S_{0\,(z)} , R_{(z)} , L_{(z)})$ in folgenden Schritten konstruieren:

1. Der Zustand z und die Elemente in $include(z)$ werden nach dem Schema in Abschnitt 7.7.15 zerlegt.
2. Für die Aktionenfolgen $en(z)$, $act_1(z)$ ... $act_k(z)$, $do(z)$ und $ex(z)$ werden nach TS1 bzw. TS2 die entsprechenden Kripke-Strukturen $KS_{en\,(z)}$, ... , $KS_{do\,(z)}$ und $KS_{ex\,(z)}$ entworfen.
3. Für $include(z)$ wird, gemäß dem in den Abschnitten 7.7.17 und 7.7.18 erläuterten Verfahren, die Kripke-Struktur $KS_{include\,(z)}$ konstruiert.
4. Die Kripke-Struktur $KS_{actions\,(z)}$ wird unter Anwendung von TS4 aus den Kripke-Strukturen $KS_{act_1\,(z)}$, ... , $KS_{act_k\,(z)}$, $KS_{do\,(z)}$ und $KS_{include\,(z)}$ gebildet, das heißt, es gilt $KS_{actions\,(z)} = (KS_{act_1\,(z)} \otimes ... \otimes KS_{act_k\,(z)} \otimes KS_{do\,(z)}) \otimes KS_{include\,(z)}$.
5. Gemäß TS3 wird aus den Kripke-Strukturen $KS_{en\,(z)}$, $KS_{actions\,(z)}$ und $KS_{ex\,(z)}$ die Kripke-Struktur $KS = (S , S_0 , R , L)$ mit $KS = KS_{en\,(z)} \oplus KS_{actions\,(z)} \oplus KS_{ex\,(z)}$ gebildet. Dann gelten:
   - 5.1. $KS_{(z)} = KS$, wenn gilt $(e, b) = (e_\varepsilon , b_\varepsilon)$.
   - 5.2. $KS_{(z)} = (S_{(z)} , S_{0\,(z)} , R_{(z)} , L_{(z)})$ mit $S_{(z)} = S$, $S_{0\,(z)} = S_0$ und $L_{(z)} = L$, wenn $(e, b) \neq (e_\varepsilon , b_\varepsilon)$ und $include(z) = \varnothing$ gelten. Für die Transitionsrelation $R_{(z)}$ gilt $R_{(z)} = R \cup (S_{actions\,(z)} \times S_{0\,ex\,(z)})$. Dabei sind
     - 5.2.1. $S_{actions\,(z)}$ die Menge der Kripke-Zustände in $KS_{actions\,(z)}$ und
     - 5.2.2. $S_{0\,ex\,(z)}$ die Menge der Anfangszustände von $KS_{ex\,(z)}$.
   - 5.3. $KS_{(z)} = (S_{(z)} , S_{0\,(z)} , R_{(z)} , L_{(z)})$ mit $S_{(z)} = S$, $S_{0\,(z)} = S_0$ und $L_{(z)} = L$, wenn $(e, b) \neq (e_\varepsilon , b_\varepsilon)$ und $include(z) \neq \varnothing$ gelten. Für $R_{(z)}$ gilt dann:

$R_{(z)} = R \cup (S_{end\_en\,(z)} \times S_{0\ en\_\ actions\,(z)}) \cup (S_{end\_ex\_\ actions\,(z)} \times S_{0\ ex\,(z)})$. Dabei sind

5.3.1. $S_{end\_en\ (z)}$ die Menge der Kripke-Zustände, die das Ende der entry-actions von z bezeichnen,

5.3.2. $S_{0\ en\_\ actions\ (z)}$ die Menge der Kripke-Zustände in $KS_{actions\ (z)}$ der Form $a_{p_1}\ a_{p_2}\ ...\ a_{p_m}$ mit der Eigenschaft, dass es eine Aktion a_{p_j} mit $1 \le j \le m$ gibt, die den Anfangszustand der entry-actions eines Teilzustands z_i von z bezeichnet, für den in Schritt 1 eine Verknüpfung zwischen $en(z)$ und $en(z_i)$ entstanden ist,

5.3.3. $S_{end\_ex\_\ actions\ (z)}$ die Menge der Kripke-Zustände in $KS_{actions\ (z)}$ der Form $a_{p_1}\ a_{p_2}\ ...\ a_{p_m}$ mit der Eigenschaft, dass jede Aktion a_{p_j} mit $1 \le j \le m$ entweder das Ende der exit-actions eines Teilzustands z_i von z bezeichnet, für den in Schritt 1 eine Verknüpfung zwischen $ex(z_i)$ und $ex(z)$ entstanden ist, oder eine Aktion der Aktionenfolgen $act_1(z)$, ... , $act_k(z)$ oder $do(z)$ ist und

5.3.4. $S_{0\ ex\,(z)}$ die Menge der Anfangszustände von $KS_{ex\,(z)}$.

In dem unter 5.1. eingeführten Fall gilt $do(z) = a_\varepsilon$, denn aus $do(z) \neq a_\varepsilon$ folgt nach der Regel 5.5.3 $(e, b) \neq (e_\varepsilon, b_\varepsilon)$.

In Schritt 5.2. verbindet bei dem Zustand z , der wegen $include(z) = \varnothing$ keine Teilzustände hat, und wegen $(e, b) \neq (e_\varepsilon, b_\varepsilon)$ seine internen Aktivitäten $act_1(z)$, ... , $act_k(z)$ und $do(z)$ unterbrochen werden können, wenn seine Austrittsbedingung erfüllt ist, die Menge $(S_{actions\,(z)} \times S_{0\ ex\,(z)})$ in $R_{(z)}$ alle internen Aktionen mit exit-actions von z . Damit werden alle möglichen Ausführungen in die Kripke-Struktur abgebildet.

In Schritt 5.3. werden für den Zustand z , der wegen $include(z) \neq \varnothing$ u. a. über seine Teilzustände aktiviert, und wegen $(e, b) \neq (e_\varepsilon, b_\varepsilon)$ auch vor dem vollständigen Abschluss seiner internen Aktivitäten verlassen werden kann, die Verbindungen zwischen den entry-actions des Oberzustands und den der Teilzustände bzw. zwischen den exit-actions der Teilzustände und den exit-actions des Oberzustands, die sich aus der Zerlegung in Schritt 1 ergeben, durch $(S_{end\_en\,(z)} \times S_{0\ en\_\ actions\,(z)})$ bzw. $(S_{end\_ex\_\ actions\,(z)} \times S_{0\ ex\,(z)})$ in die Transitionsrelation $R_{(z)}$ der Kripke-Struktur aufgenommen.

Mit Hilfe von TS5 kann jetzt ein Zustandsdiagramm bzw. eine Zustandsmaschine der UML in eine Kripke-Struktur transformiert werden.

7.7.23 Transformation eines Zustandsdiagramms

Es sei $\mathcal{M}_{K^i} = (\mathcal{Z}, \mathcal{T}, default(), final())$ das Zustandsdiagramm bzw. die Zustandsmaschine der Klasse K^i. Die Transformation von $\mathcal{M}_{K^i}$ erfolgt in den folgenden Schritten:

1. Unter der Anwendung von TS5 wird für jeden Zustand in $\mathcal{Z}$ eine Kripke-Struktur entworfen.
2. Für jede Transition in $\mathcal{T}$ wird, wie in Abschnitt 7.7.14 gezeigt wurde, eine Kripke-Struktur konstruiert.
3. Beginnend mit der Kripke-Struktur der Initialtransition werden die Kripke-Strukturen aller Zustände und die ihrer Transitionen unter Einhaltung der von den Transitionen gegebenen Reihenfolge sequenziell verknüpft.

Damit wird das in dem objektorientierten Modell entworfene Verhalten der Systemkomponenten in geeignete Kripke-Strukturen transformiert und so für die Durchführung einer formalen Verifikation vorbereitet.

Wie Abb. 7.1 zeigt, müssen noch die Prüfbedingungen, die in der Regel natürlich sprachlich vorliegen, als temporallogische Ausdrücke formuliert werden, damit deren Gültigkeit im Modell überprüft werden kann.

7.8 Logiksprachen

Anhand der Logiksprachen können Prüfbedingungen für die Durchführung der formalen Verifikation formuliert werden. Diese Prüfbedingungen sind Aussagen über die funktionalen Anforderungen an das System. Für den Aufbau und die Funktionsweise der Logiksprachen sei auf [HR00] verwiesen. Hier wird die Logiksprache Computation Tree Logic (CTL) vorgestellt, die von SMV erkannt und bearbeitet werden kann.

7.8.1 Computation Tree Logic

1981 wurde die Computation Tree Logic (CTL) in [CE81a, CE81b] vorgestellt. Sind die Zustände, die ein System in seinem Lebenszyklus einnimmt, in Form eines Baums modelliert, so können mit Hilfe von CTL Aussagen über Zustände dieses Baums formuliert werden.

In vorangegangenen Abschnitten wurde gezeigt, dass das dynamische Verhalten von Systemkomponenten, das in Form von Zustandsdiagrammen der UML vorliegt, sich in Kripke-Strukturen, also in Form von verzweigten Bäumen, darstellen lässt. Dadurch ist

es möglich, mit Hilfe von CTL Aussagen über dieses Verhalten zu formulieren. Diese Aussagen können funktionale Anforderungen an die Systemkomponente sein.

Wie jede andere Logiksprache verfügt CTL über eine eindeutige und strenge Syntax und Semantik. Diese sind mit einigen Unterschieden in der Notation und Darstellung in [HR00, CGP00, Pin02] beschrieben. Hier werden die Syntax und Semantik von CTL in Anlehnung an [HR00] präsentiert.

CTL-Ausdrücke werden über baumartige Strukturen wie z. B. Kripke-Strukturen definiert, die in Abschnitt 7.5 eingeführt wurden. Bezogen auf diese Strukturen und CTL-Ausdrücke werden die Syntax und die Semantik für einen CTL-Ausdruck definiert.

7.8.1.1 Syntax von CTL (Definition)

Ein Ausdruck ϕ in CTL besteht aus einer Zusammensetzung von Formeln ϕ bzw. atomaren Aussagen p. Die Formeln selbst sind wiederum eine Zusammensetzung von atomaren Aussagen. Formal lässt sich die Syntax einer CTL-Formel, wie folgt, definieren:

$$\phi ::= \bot \mid \top \mid p \mid (\neg\phi) \mid (\phi \wedge \phi) \mid (\phi \vee \phi) \mid (\phi \Rightarrow \phi) \mid \mathbf{AX}\ \phi \mid \mathbf{EX}\ \phi \mid \mathbf{A}\ [\phi\ \mathbf{U}\ \phi] \mid \mathbf{E}\ [\phi\ \mathbf{U}\ \phi]$$
$$\mid \mathbf{AG}\ \phi \mid \mathbf{EG}\ \phi \mid \mathbf{AF}\ \phi \mid \mathbf{EF}\ \phi$$

In dieser Darstellung steht das Symbol „ | “ für „ oder “. Das Symbol „ $\bot$ “ zeigt eine Formel, deren Wahrheitswert immer falsch ist. Ebenfalls steht das Symbol „$\top$ “ für eine CTL-Formel, deren Wahrheitswert immer richtig ist. Beispiele für solche Formeln sind ($p \wedge \neg p$) bzw. ($p \vee \neg p$). Die Ausdrücke ($\neg\phi$), ($\phi \wedge \phi$), ($\phi \vee \phi$) und ($\phi \Rightarrow \phi$) sind CTL-Formeln, wenn an der Stelle von ϕ eine syntaktisch korrekte CTL-Formel steht.

Die Symbole „**A**“ (alle Pfade entlang) bzw. „**E**“ (wenigstens einen Pfad entlang) kommen immer mit einem der Symbole „**X**“ (nächster Zustand), „**F**“ (irgendein Zustand in Zukunft), „**G**“ (alle zukünftigen Zustände) oder „**U**“ vor und umgekehrt, das heißt, jedes Symbol „**X**“, „**F**“, „**G**“ oder „**U**“ tritt nur als Paar mit einem der Symbole „**A**“ bzw. „**E**“ auf. Demnach sind die korrekten Schreibweisen nur die oben angegebenen.

Für „**U**“ gilt eine der Schreibweisen „**A** [ϕ_1 **U** ϕ_2]“ bzw. „**E** [ϕ_1 **U** ϕ_2]“ als ein Paar. Diese stehen für „alle Pfade entlang“ bzw. „wenigstens einen Pfad entlang“ gilt ϕ_1 bis (bevor) ϕ_2 gilt.

Als Pfad gilt eine Reihe von Zuständen, die paarweise über Transitionen verbunden sind. Entsprechend der Definition 7.5 ist $\{(s_i, s_{i+1}), (s_{i+1}, s_{i+2}), \ldots, (s_{j-1}, s_j)\} \subseteq R$ ein Pfad von s_i nach s_j in *KS*.

Mit der Syntax für CTL wird die korrekte Schreibweise für die CTL-Ausdrücke festgelegt. Die Bedeutung von CTL-Ausdrücken wird durch die Semantik für einen CTL-Ausdruck definiert.

7.8.1.2 Semantik von CTL (Definition)

Es seien $KS = (S, S_0, R, L)$ eine Kripke-Struktur über $V = \{ p, q, ... \}$ eine Menge atomarer Aussagen und $L(s_i) \subseteq V$ die Menge aller atomaren Aussagen, die in $s_i \in S$ gelten. Dass ein CTL-Ausdruck ϕ in einem Zustand $s_i \in S$ von KS gilt, wird mit der Schreibweise $KS, s_i \vDash \phi$ bezeichnet. Das ist dann erfüllt, wenn einer der folgenden Fälle vorliegt:

1. $KS, s_i \vDash \top$ und $KS, s_i \nvDash \bot$ für alle $s_i \in S$ gelten, das heißt, die immer gültigen Aussagen gelten in jedem Zustand und die immer ungültigen Aussagen gelten in keinem Zustand.

2. $KS, s_i \vDash p$ gilt genau dann, wenn $p \in L(s_i)$ gilt, das heißt, in jedem Zustand gelten nur die Aussagen, die für diesen Zustand als gültig gelten.

3. $KS, s_i \vDash \neg\phi$ gilt genau dann, wenn $KS, s_i \nvDash \phi$ gilt, das heißt, wenn eine CTL-Formel in einem Zustand gilt, so gilt ihr Gegenteil nicht in demselben Zustand.

4. $KS, s_i \vDash \phi_1 \wedge \phi_2$ gilt genau dann, wenn $KS, s_i \vDash \phi_1$ und $KS, s_i \vDash \phi_2$ gelten, das heißt, eine Konjunktion zweier CTL-Formeln gilt in einem Zustand, wenn die einzelnen CTL-Formeln in demselben Zustand gelten und umgekehrt.

5. $KS, s_i \vDash \phi_1 \vee \phi_2$ gilt genau dann, wenn $KS, s_i \vDash \phi_1$ oder $KS, s_i \vDash \phi_2$ gelten, das heißt, eine Disjunktion zweier CTL-Formeln gilt in einem Zustand, wenn mindestens eine von ihnen in demselben Zustand gilt und umgekehrt.

6. $KS, s_i \vDash \phi_1 \Rightarrow \phi_2$ gilt genau dann, wenn $KS, s_i \nvDash \phi_1$ oder $KS, s_i \vDash \phi_2$ gelten, das heißt, die Implikation $\phi_1 \Rightarrow \phi_2$ zweier CTL-Formeln ϕ_1 und ϕ_2 gilt in einem Zustand, wenn in dem Zustand ϕ_1 nicht gilt. Ansonsten muss ϕ_2 gelten und umgekehrt.

7. $KS, s_i \vDash$ **AX** ϕ gilt genau dann, wenn für alle Zustände s_j mit $(s_i, s_j) \in R$ die Aussage $KS, s_j \vDash \phi$ gilt. Daher bedeutet **AX** „in allen Folgezuständen gilt ϕ“ bzw. „in allen nächsten Zuständen gilt ϕ“.

8. $KS, s_i \vDash$ **EX** ϕ gilt genau dann, wenn es irgendeinen Zustand s_j mit $(s_i, s_j) \in R$ und $KS, s_j \vDash \phi$ gibt. Daher bedeutet **EX** „in irgendeinem der Folgezustände gilt ϕ“ bzw. „in irgendeinem nächsten Zustand gilt ϕ“.

9. $KS, s_i \models$ **AG** ϕ gilt genau dann, wenn bei allen Ausführungspfaden $\{ (s_i , s_{i+1}), (s_{i+1} , s_{i+2}), ... \} \subseteq R$ die Aussage $KS, s_j \models \phi$ für alle $s_j \in \{ s_i , s_{i+1} , ... \}$ gilt. Daher bedeutet **AG** „auf allen Pfaden und in allen Zuständen gilt ϕ“.

10. $KS, s_i \models$ **EG** ϕ genau dann gilt, wenn es einen Ausführungspfad $\{ (s_i , s_{i+1}), (s_{i+1} , s_{i+2}), ... \} \subseteq R$ gibt, so dass $KS, s_j \models \phi$ für alle $s_j \in \{ s_i , s_{i+1} , ... \}$ gilt. Daher bedeutet **EG** „auf irgendeinem Pfad aber in allen Zuständen gilt ϕ“.

11. $KS, s \models$ **AF** ϕ gilt genau dann, wenn bei allen Pfaden $\{ (s_i , s_{i+1}), (s_{i+1} , s_{i+2}), ... \} \subseteq R$ die Aussage $KS, s_j \models \phi$ für irgendeinen $s_j \in \{ s_i , s_{i+1} , ...\}$ gilt. Daher bedeutet **AF** „auf allen Pfaden gilt ϕ in irgendeinem Zustand“.

12. $KS, s_i \models$ **EF** ϕ gilt genau dann, wenn ein Pfad $\{ (s_i , s_{i+1}), (s_{i+1} , s_{i+2}), ... \} \subseteq R$ existiert, so dass $KS, s_j \models \phi$ für irgendeinen $s_j \in \{ s_i , s_{i+1} , ... \}$ gilt. Daher bedeutet **EF** „auf irgendeinem Pfad und in irgendeinem Zustand gilt ϕ“.

13. $KS, s_i \models$ **A** $[\phi_1$ **U** $\phi_2]$ gilt genau dann, wenn der Ausdruck ϕ_1 **U** ϕ_2 bei allen Pfaden $\{ (s_i , s_{i+1}), (s_{i+1} , s_{i+2}), ... \} \subseteq R$ erfüllt ist, das heißt, jeden Pfad entlang gibt es einen $s_k \in \{ s_i , s_{i+1} , ... \}$ mit $KS, s_k \models \phi_2$ und für alle $s_j \in \{ s_i , s_{i+1} , ... \}$ mit $j < k$ gilt $KS, s_j \models \phi_1$. Daher bedeutet **A** $[\phi_1$ **U** $\phi_2]$ „in allen Zuständen gilt ϕ_1 bis (bevor) ϕ_2 gilt“.

14. $KS, s_i \models$ **E** $[\phi_1$ **U** $\phi_2]$ gilt genau dann, wenn es in der Transitionsrelation irgendeinen Pfad $\{ (s_i , s_{i+1}), (s_{i+1} , s_{i+2}), ... \} \subseteq R$ mit der unter 13 genannten Eigenschaft gibt.

In dieser Definition bedeutet der Ausdruck „ p genau dann, wenn q “, dass p aus q und q aus p folgen.

Unter Anwendung der eingeführten Syntax und Semantik werden die zu überprüfenden Systemeigenschaften in Form von CTL-Formeln niedergeschrieben und damit für die Durchführung des Model-Checking vorbereitet.

7.9 Übertragung von Prüfbedingungen in CTL-Formeln

In Kapitel drei wurde erwähnt, dass die Systemanforderungsspezifikation gegenüber Sicherheitsanforderungen bzw. bezüglich der gestellten funktionalen Anforderungen überprüft werden soll.

Bei der Anwendung der Technik Model-Checking mit SMV müssen die zu überprüfenden Systemeigenschaften, die hier Prüfbedingungen genannt werden, in CTL formuliert worden sein.

Eine Reihe von Prüfbedingungen ergibt sich z. B. direkt aus den Normen, Gesetzen und Eisenbahnvorschriften, oder entsteht durch Anforderungen an Systemkomponenten hinsichtlich ihrer physikalischen Eigenschaften für eine sichere Betriebsführung und für die Vermeidung von Gefährdungen. Derartige Prüfbedingungen liegen in der Regel in natürlich sprachlich formulierter Form vor.

Eine weitere Gruppe von Prüfbedingungen resultiert aus den Sequenzdiagrammen, die bei der Erstellung des Modells für die Modellierung der Interaktionen unter den Systemkomponenten entworfen worden sind.

7.9.1 Übertragung von natürlich sprachlichen Prüfbedingungen in CTL-Formeln

In Abschnitt 3.2.2 wurden einige natürlich sprachlich formulierte Sicherheitsanforderungen an den FFB gezeigt. Hier wird am Beispiel der ersten Anforderung die Umformulierung in eine CTL-Formel dargestellt.

Die Sicherheitsanforderung lautet: „Bei einer technischen Sicherungsanlage darf der Sicherungsvorgang nicht zu spät begonnen und nicht zu früh beendet werden“.

Um diese ganz allgemein formulierte Anforderung in eine geeignete CTL-Formel umzuschreiben, müssen zuerst die Objekte identifiziert werden, die diese Anforderung erfüllen müssen. In dem objektorientierten Modell ist der eingleisige Bahnübergang das Objekt, das als eine „technische Sicherungsanlage“ diese Sicherheitsanforderung zu erfüllen hat.

Im nächsten Schritt muss bei dem zutreffenden Objekt die Aufgabe des „Sicherungsvorgangs“ konkretisiert werden. Der Sicherungsvorgang eines technisch gesicherten eingleisigen Bahnübergangs besteht aus dem Herstellen einer Sperrung des Gefahrenraums, bevor der Gefahrenraum von einem Eisenbahnfahrzeug betreten wird, und dem Aufheben der Sperrung, nachdem das Eisenbahnfahrzeug den Gefahrenraum geräumt hat. Für die Bahnübergänge im FFB kann ein „zu spätes“ Beginnen des Sicherungsvorgangs keine sicherheitsverletzenden Folgen haben. Denn durch die Konzeption des Betriebsablaufs darf ein Bahnübergang nicht befahren werden, bevor das Fahrzeug die Quittung über den gesicherten Zustand des Bahnübergangs erhalten hat. Das Beenden des Sicherungsvorgangs erfolgt bei einem technisch gesicherten eingleisigen Bahnübergang, der mit Lichtzeichen- und Schrankenanlage ausgerüstet ist (Kapitel drei), mit dem Ausschalten der Lichtzeichenanlage. Dieser Vorgang wurde in dem Zustandsdiagramm der Klasse *Bahnübergang* durch die Aktion *lzA_ausschalten*() abgebildet. Damit wird die Aktion in dem entsprechenden Zustandsdiagramm identifiziert, die für die genannte Anforderung relevant ist. Diese Aktion entspricht dem gleichnamigen Zustand in der

Kripke-Struktur der Klasse *Bahnübergang*, an der die tatsächliche Überprüfung bzw. die formale Verifikation durch SMV erfolgt.

In Kapitel drei wurde dargestellt, dass der Sicherungsvorgang beendet wird, wenn die Zugfahrt stattgefunden hat bzw. die Grundstellungszeit (t_{gs}) abgelaufen ist. Beim Ablaufen von t_{gs} sendet der Bahnübergang eine Meldung über die Zeitüberschreitung an die Zentrale.

Eine Zugfahrt wird von der Klasse *Bahnübergang* erkannt, wenn die Klasse *Ausschaltsensorik* die Aktion *sen_melden_frei*() ausführt, nachdem sie die Aktion *sen_melden_besetzt*() ausgeführt hat. Daraus folgt die Teilformel, die die Umstände für ein reguläres Beenden des Sicherungsvorgangs ausdrückt:

„*Bahnübergang.zeit_melden*(t_{gs}) |

Ausschaltsensorik.sen_melden(*besetzt*) ⇒ *Ausschaltsensorik.sen_melden*(*frei*)“. (1)

Ein „zu frühes“ Beenden des Sicherungsvorgangs heißt folglich das Ausschalten der Lichtzeichenanlage, bevor diese Zustände erreicht werden. Dies wird durch die Verneinung von *lzA_ausschalten* in allen vorangehenden Zuständen gefordert.

„**A** (*Bahnübergang.*¬ *lzA_ausschalten*() **U**

(*Bahnübergang.zeit_melden*(t_{gs}) |

(*Ausschaltsensorik.sen_melden*(*besetzt*) ⇒ *Ausschaltsensorik.sen_melden*(*frei*))))“. (2)

In (2) zeigt **A** ... **U**, dass (*Bahnübergang.*¬ *lzA_ausschalten*()) auf allen Pfaden und in allen Zuständen gilt, bevor (1) gilt, das heißt, „*lzA_ausschalten*()“ gilt nicht, bis die Zustände erreicht sind, in denen die Voraussetzungen für das Beenden des Sicherungsvorgangs erfüllt sind.

Diese Anforderung muss allerdings dann gelten, wenn der Sicherungsvorgang bereits angefangen wurde, das heißt, (1) soll nach dem Zustand gelten, in dem die Aktion *lzA_einschalten*() stattgefunden hat. Damit ergibt sich die folgende Erweiterung:

„*Bahnübergang.lzA_einschalten*() ⇒

A (*Bahnübergang.*¬ *lzA_ausschalten*() **U**

(*Bahnübergang.zeit_melden*(t_{gs}) |

(*Ausschaltsensorik.sen_melden*(*besetzt*) ⇒ *Ausschaltsensorik.sen_melden*(*frei*))))“ (3)

Die letzte Vervollständigung erfährt die Teilformel (3), indem sie mit **AG** erweitert wird, damit sie immer auf allen Pfaden und in allen Zuständen des Modells gilt, wenn der Sicherungsvorgang beginnt:

AG (*Bahnübergang.lzA_einschalten*() ⇒

A (*Bahnübergang.*¬ *lzA_ausschalten*() **U**

(*Bahnübergang.zeit_melden*(t_{gs}) |

(*Ausschaltsensorik.sen_melden*(*besetzt*) ⇒ *Ausschaltsensorik.sen_melden*(*frei*))))) (4)

Die CTL-Formel (4) ist nun eine Prüfbedingung für das objektorientierte Modell.

7.9.2 Übertragung von Sequenzdiagrammen in CTL-Formeln

Die Interaktionen zwischen den Systemkomponenten sind eine weitere Gruppe der funktionalen Anforderungen, die als Prüfbedingungen gestellt werden können. Dabei kann untersucht werden, ob das lokale in Zustandsdiagrammen definierte Verhalten von Komponenten die im Rahmen des Gesamtsystems erforderliche Aufgabe erfüllt.

In Kapitel fünf wurde gezeigt, dass die Interaktionen unter den Systemkomponenten in Szenarios modelliert werden, die mittels Sequenzdiagramme der UML beschrieben werden. Für jede zwei Interaktionen (O_i .[b] a , O_j , p) und (O_k .[b'] a' , O_r , p') eines Szenarios lässt sich über die in Abschnitt 5.13 eingeführte Ordnungsrelation eine logische Implikation definieren. Dabei gilt

O_i .[b] $a \Rightarrow O_k$.[b'] a', wenn (O_i .[b] a , O_j , p) $\preccurlyeq$ (O_k .[b'] a' , O_r , p') gilt.

Das bedeutet, die Aktion a wird unter der Bedingung b von dem Objekt O_i ausgeführt, danach bzw. gleichzeitig wird die Aktion a' unter der Bedingung b' von dem Objekt O_k ausgeführt.

Die Bedingungen b und b', die sich von b_ε unterscheiden, werden unter Anwendung von Negation, Konjunktion, Disjunktion bzw. Implikation in logische Teilformeln umgewandelt. Die Bedingung b_ε entspricht der Formel $\top$, die laut Semantik von CTL in jedem Zustand aller Kripke-Strukturen erfüllt ist. Schließlich wird die Formel mit **AG**, **EG**, **AF** etc. vervollständigt, wenn die Interaktion immer vorkommen muss, vorkommen darf etc.

Abb. 7.17 zeigt ein Szenario mit den Interaktionen zwischen einem Objekt der Klasse *Fahrzeug* und einem Objekt der Klasse *FFB-Zentrale*. Dieses Szenario beschreibt einen Kommunikationsablauf zwischen dem Fahrzeug und der FFB-Zentrale vor einer Abfahrt des Fahrzeugs. Demnach fordert das Fahrzeug zuerst bei der FFB-Zentrale eine Fahrerlaubnis (ferl) an. Danach wird von der FFB-Zentrale unter der Bedingung, dass der angeforderte Fahrweg frei ist, eine Fahrerlaubnis an das Fahrzeug gesendet. Erst nach der Erteilung der Fahrerlaubnis löst das Fahrzeug seine Bremsen und fährt los. Die Abfahrt selbst ist allerdings von der Bedingung abhängig, dass die Abfahrtzeit (t_a) erreicht ist, das heißt,, dass die Aktuelle Zeit (t_{akt}) und t_a gleich sind.

Dieses Szenario entspricht der in Abschnitt 3.2.2 genannten Sicherheitsanforderung, dass ein Zug nur abfahren darf, wenn ihm von der FFB-Zentrale ein Fahrweg (enthalten in einer Fahrerlaubnis) zugewiesen worden ist.

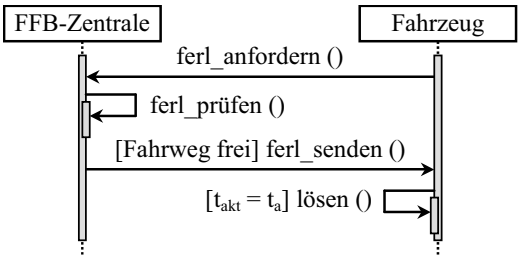


Abb. 7.17 Kommunikation zwischen dem Fahrzeug und der FFB-Zentrale vor einer Abfahrt

An diesem Szenario sind drei Interaktionen „*Fahrzeug.ferl_anfordern*()“, „*FFB-Zentrale.*[*Fahrweg frei*] *ferl_senden*()“ und „*Fahrzeug.*[$t_{akt} = t_a$] *lösen*()“ beteiligt.

Die ersten zwei Interaktionen initiieren die folgende Implikation:

Fahrzeug.ferl_anfordern() $\Rightarrow$ *FFB-Zentrale.*[*Fahrweg frei*] *ferl_senden*().

Unter Anwendung der Konjunktion ergibt sich:

Fahrzeug.ferl_anfordern() $\Rightarrow$ *FFB-Zentrale.*((*Fahrweg = frei*) $\wedge$ *ferl_senden*()).

Die dritte Interaktion *Fahrzeug.*[$t_{akt} = t_a$] *lösen*() darf nicht zwischen den ersten zwei Interaktionen vorkommen, damit die Abfahrt des Fahrzeugs nicht vor der Erteilung der Fahrerlaubnis stattfindet. Daher ergibt sich:

Fahrzeug.ferl_anfordern() $\Rightarrow$

A (*Fahrzeug.* $\neg$(($t_{akt} = t_a$) $\wedge$ *lösen*()))**U***FFB-Zentrale.*((*Fahrweg = frei*) $\wedge$ *ferl_senden*())).

Da diese Formel immer in der Systemanforderungsspezifikation erfüllt sein muss, wird die Formel mit **AG**, wie folgt, vervollständigt:

AG (*Fahrzeug.ferl_anfordern*() $\Rightarrow$

A (*Fahrzeug.* $\neg$(($t_{akt} = t_a$) $\wedge$ *lösen*()))**U***FFB-Zentrale.*((*Fahrweg = frei*) $\wedge$ *ferl_senden*())).

7.10 Verifikation des objektorientierten Modells

Nachdem die Zustandsdiagramme des objektorientierten Modells in geeignete Kripke-Strukturen transformiert und die Prüfbedingungen in CTL-Formeln umgeschrieben wurden, werden diese in die Eingangssprache von SMV implementiert.

Für die Verifikation von objektorientierten Systemen wird in SMV die Kripke-Struktur jedes Objekts in Form eines eigenständigen Moduls implementiert. Für die Interaktionen unter den Objekten existieren jedoch keine speziellen Sprachkonstrukte, die für die Abbildung der Methodenaufrufe bzw. für die Verknüpfung zwischen aufrufendem Er-

eignis und aufgerufener Aktion verwendet werden können. Daher müssen für die Kommunikation zwischen diesen Objekten gemeinsame Variablen definiert werden.

Die Module der einzelnen Objekte kommunizieren über ein Hauptmodul. In diesem Hauptmodul werden Variablen definiert, auf die alle Module zugreifen können. Gilt eine Aktion a_i aus einem Modul m_i als ein Ereignis für ein Modul m_j , worauf in m_j die Aktion a_j ausgeführt wird bzw. ein Zustandswechsel stattfindet, so wird in dem Hauptmodul eine entsprechende Variable definiert, deren Wert mit der Ausführung von a_i geändert wird. Das Modul m_j wartet bis diese Variable den erwarteten Wert annimmt. Danach führt m_j die Aktion a_j aus bzw. ändert seinen aktuellen Zustand. Die Implementierung des objektorientierten Modells des Anwendungsbeispiels wird in [Kuz04] ausführlich erläutert.

Bei der Verifikation dieses Modells, das aus 17 Modulen für 17 Objekte des Anwendungsbeispiels und einem Hauptmodul besteht, wurde das Modell zuerst nach Erreichbarkeit untersucht. Gegenstand dieser Untersuchung ist, ob alle Kripke-Zustände erreicht werden bzw. alle Aktionen ausgeführt werden. In einer weiteren Analyse wurde das Modell nach Lebendigkeit untersucht. Dabei geht es darum, ob jeder Zustand auch verlassen wird. Durch diese Eigenschaften wurde die grundsätzliche Korrektheit des Modells festgestellt. Danach wurde das Modell gegenüber Sicherheitsanforderungen und gegenüber funktionalen Anforderungen, die in Sequenzdiagrammen beschrieben wurden, untersucht.

In diesem Kapitel wurde der Weg von einer objektorientierten Systemanforderungsspezifikation zur formalen Verifikation durch die Anwendung der Technik Model-Checking gezeigt. Mit einer kleinen Diskussion über die Probleme mit der Technik des Model-Checking wird dieses Kapitel abgeschlossen.

7.11 Grenzen des Model-Checking

Model-Ckecking ist eine effiziente Technik zur Durchführung formaler Verifikation mit Beweischarakter. Durch die Anwendendung dieser Technik kann mehr Vertrauen in eine Systemanforderungsspezifikation gewonnen werden, als durch manuelle Untersuchungen von natürlich sprachlich verfassten Lastenheften. Dennoch ist bei der Anwendung von Model-Checking mit zwei Problemen zu rechnen.

Das eine Problem ist das sogenannte Zustandsexplosionsproblem. Mit diesem Begriff wird das Wachstum des zu verifizierenden Zustandsraums bezeichnet, das eine inhärente Eigenschaft von komplexen Systemen ist. Das Wachstum des Zustandsraums rührt vor allem daher, dass sich bei Systemen mit mehreren nebenläufigen Prozessen der gesamte Zustandsraum aus der Kombination der Zustände einzelner Komponenten bildet. Für n Prozesse mit jeweils durchschnittlich p Zuständen und q Variablen entspricht die Größe des Zustandsraums, der aus allen möglichen Kombinationen von Variablen be-

steht, n^{p^q} Zuständen. Dieses Problem führt dazu, dass die zu verifizierenden Strukturen nicht mehr mit einem vertretbaren Hardwareaufwand abgebildet und analysiert werden können. Um dem exponentiellen Wachstum des Zustandsraums entgegenzuwirken, wird eine Reihe von Ansätzen verfolgt.

Ein Weg zur Reduzierung des Zustandsraums bietet der sogenannte Kompositionalitätsansatz [OG76, CLM89]. Bei diesem Ansatz werden die nebenläufigen Prozesse nicht miteinander kombiniert. Jede Systemkomponente wird separat analysiert. Dabei müssen zu jeder einzelnen Systemkomponente noch die Interaktionen mit den anderen Systemkomponenten oder mit der Systemumgebung mitmodelliert werden. Demzufolge werden die Interaktionen mehrfach beschrieben, wobei die mehrfache Beschreibung in sich konsistent sein muss. Nach dem Abschluss des Verifikationsprozesses wird angenommen, dass das lokal verifizierte Verhalten auch im globalen Zustandsraum gültig ist.

Eine weitere Gruppe der Ansätze bilden die sogenannten Reduktionsverfahren. Diese Ansätze beschäftigen sich damit, den globalen Zustandsraum zu reduzieren. Zu dieser Gruppe gehört die Halbordnungsreduktion [KP88, Val90, CGP00]. Dabei handelt es sich um eine Reduzierung der parallelen Zustandsübergänge, die ihrerseits zur Reduzierung des globalen Zustandsraums führen.

Mit dem Begriff Abstraktion [CGL92] identifiziert sich eine weitere Reihe von Verfahren, die die Reduzierung des globalen Zustandsraums dadurch ermöglichen, dass sie in den Verifikationsprozess lediglich die Teile des Systems einschließen, die für die vorliegenden Prüfbedingung relevant sind.

Ebenso tragen die Symmetrieverfahren [ES93, CFJ93, CGP00] zur Reduzierung des globalen Zustandsraums bei. Dabei geht es um die Bildung von Äquivalenzklassen, die Systemkomponenten mit identischem dynamischem Verhalten umfassen. Damit wird der globale Zustandsraum drastisch reduziert.

Ein weiterer Ansatz zur effektiven Reduzierung des Verifikationsaufwands in einer objektorientierten Anwendung ist der Einsatz von Multi-Objekt Logiken zur Formalisierung von Prüfbedingungen. Dabei wird jede Prüfbedingung, die Aussagen über mehrere Objekte enthält, in Teilspezifikationen zerlegt, die aus der Sicht der einzelnen Objekte formuliert worden sind. In diesem Verfahren wird durch die Verifikation der einzelnen Objekte gegenüber Teilspezifikationen das Wachstum des Zustandsraums verhindert und der Verifikationsaufwand drastisch reduziert. [Pin02, EKP02, EKP03] stellen das vollautomatische Verifikationsverfahren Multi-Object-Checking vor. Dabei werden für eine in der Multiobjekt-Logik D_1 [EC+98, EP00, EC00] formulierte Spezifikation eine Zerlegung in Teilspezifikationen in D_0 [EC00] vorgenommen wird. Dadurch wird nicht mehr für die Überprüfung der einzelnen Teilspezifikationen der gesamte Zustandsraum benötigt.

Die weiteren Ansätze zur Reduzierung des globalen Zustandsraums werden in [Pin02] diskutiert. Dort wird auch weiterführende Literatur genannt.

Das zweite Problem mit formaler Verifikation besteht darin, dass die Formulierung der Prüfbedingungen in eine Logiksprache schwierig und meist fehleranfällig ist. Um dieser Schwierigkeit entgegenzuwirken, verfolgen einige Ansätze [Ort97, Sch97, FMR00] den Weg über eine strukturierte Sprache bzw. eine Normsprache. Diese Sprachen basieren auf den natürlichen Sprachen und verwenden eine genormte Terminologie bzw. Begriffskonvention. Dabei werden für einige ausgewählte Begriffe aus einer natürlichen Sprache fundierte Semantiken definiert. Durch diese Semantiken werden die Begriffe einer Normsprache unabhängig vom Anwendungskontext. Im Vergleich zu natürlichen Sprachen ist eine Normsprache klarer und präziser. Folglich erleichtern die fundierten Semantiken der Begriffe den Schritt von einer natürlich sprachlichen zu einer formalen Formulierung in einer Logiksprache.

Ein weiterer Ansatz ist die Idee mit Safety-Patterns [Bit01, BG02, ABG04]. Dieser Ansatz geht von der These aus, dass eine reduzierte Ausdrucksmächtigkeit von Temporallogik für die formale Spezifikation funktionaler Sicherheitsanforderungen für Automatisierungssysteme in der Praxis ausreicht. Diese reduzierte Ausdrucksmächtigkeit wird auf eine endliche Anzahl von Pattern abgebildet. Safety-Pattern beinhalten nur genau die temporallogischen Ausdrücke und entsprechende natürlich sprachliche Formulierungen, die für die Spezifikation der verschiedenen Arten funktionaler Sicherheitsanforderungen gebraucht werden.

8 Ergebnisse der Verifikation

Im folgenden Kapitel wird der Nutzen der formalen Verifikation anhand von Fehlern demonstriert, die sich in einer natürlich sprachlich verfassten Systemanforderungsspezifikation äußerst schwierig entdecken lassen. Durch die Anwendung der Technik Model-Checking erfolgt die Suche nach Fehlern in der Systemspezifikation automatisch. Die hier demonstrierten Fehler beziehen sich auf das Zustandsdiagramm der Klasse *Bahnübergang* (Abb. 3.6), das das dynamische Verhalten des eingleisigen Bahnübergangs im FFB spezifiziert.

Für die formale Überprüfung des objektorientierten Modells, das in der Notation der UML vorliegt, werden zuerst die Zustandsdiagramme des Modells, wie bereits in Kapitel sieben gezeigt wurde, in geeignete Kripke-Strukturen transformiert, die in die Eingangssprache von SMV implementiert werden. Parallel dazu werden die Ziele festgelegt, die in jedem Verifikationsschritt verfolgt werden. Entsprechend den festgelegten Zielen werden adäquate Prüfbedingungen aufgestellt und in temporallogischen Formeln formuliert. Der Model-Checker überprüft dann automatisch die Korrektheit der vorgelegten Kripke-Struktur gegenüber den Prüfbedingungen. Wird nach der Überprüfung kein Fehler ausgegeben, erfüllt das Model sämtliche gestellte Prüfbedingungen. In Fehlerfällen wird von dem Model-Checker für jede Prüfbedingung jeweils ein Pfad ausgegeben, in dem die Prüfbedingung verletzt wird. Der ausgegebene Pfad wird dann für die Korrektur des Modells eingesetzt. In den nächsten Abschnitten wird auf diese Schritte näher eingegangen.

8.1 Transformation des Zustandsdiagramms der Klasse *Bahnübergang*

Das Zustandsdiagramm der Klasse *Bahnübergang* (Abb. 3.6) besteht aus drei Zuständen *Automatischer_Betrieb*, *Örtlich_bedienbar* und *Defekt*. Für diese Zustände und für ihre Transitionen werden unter Anwendung der Transformationsschemata und Verfahren, die in Kapitel sieben eingeführt wurden, Kripke-Strukturen entworfen, aus denen sich die Kripke-Struktur des gesamten Zustandsdiagramms konstruieren lässt.

Die Initialtransition wird von dem Ereignis *inBetriebNehmen*(*bü*) gesteuert. Dieses Ereignis löst die Ausführung der Aktionenfolge *inBetriebNehmen*(*tim*), *inBetriebNehmen*(*kom*), *inBetriebNehmen*(*lzA*), *inBetriebNehmen*(*schA*), *inBetriebNehmen*(*sen*) aus. Die Kripke-Struktur dieser Aktionenfolge lässt sich gemäß TS1 (Transformationsschema 1) bilden. Nach der Ausführung dieser Aktionenfolge wird der Zustand *Automatischer_Betrieb* aktiviert. Dieser Zustand besteht aus zwei parallelen Regionen. Zuerst werden die einzelnen Teilzustände *Testend*, *Frei*, *Auftrag_Bearbeitend*, etc. dieser Regionen entsprechend dem in Abschnitt 7.7.15 erläuterten Verfahren zerlegt. Danach werden die Teilzustände dieser Regionen, die sich nach der Zerlegung ergeben, gemäß Abschnitt 7.7.19 kombiniert, und ihre Transitionen werden auf die kombinierten Zustände übertragen. Die neuen Zustände haben kombinierte Zustandsnamen *Frei Testend*, *Auftrag_Bearbeitend Testend*, etc. In diesen Zuständen werden mit Hilfe von TS1 für die Aktionenfolgen der beiden Teilzustände Kripke-Strukturen entworfen, die dann parallel verknüpft werden. Anschließend werden die entstandenen Kripke-Strukturen sequenziell verknüpft.

Aus den sequenziellen Verknüpfungen der Kripke-Strukturen dieses Zustands mit den Kripke-Strukturen der Zustände *Defekt*, *Örtlich_bedienbar* und der Kripke-Struktur der Initial- und Finaltransition lässt sich dann eine Kripke-Struktur für das gesamte Zustandsdiagramm der Klasse *Bahnübergang* konstruieren.

8.2 Verifikationsziele und entsprechende Prüfbedingungen

Eine Kripke-Struktur kann auf unterschiedliche Bedingungen bezüglich des dynamischen Verhaltens der modellierten Komponente überprüft werden.

Bei der Modellierung des dynamischen Verhaltens einer Systemkomponente ist zu untersuchen, ob während des Lebenszyklus der Komponente alle modellierten Zustände erreicht und alle Aktionen ausgeführt werden. Für die Objekte der Klasse *Bahnübergang* heißt es, ob z. B. der Zustand erreicht wird, in dem die Aktion *quittieren*() ausgeführt wird. Diese Bedingung wird in temporallogischer Formel „**AG** ($a_0 \Rightarrow$ **AF** *quittieren*())“ ausgedrückt. Diese Formel verlangt, dass auf allen Pfaden, die von dem Kripke-Zustand a_0 ausgehen, *quittieren*() vorkommt. Dabei ist a_0 der Kripke-Zustand, der ge-

mäß TS1 (Abschnitt 7.7.2) bei der Konstruktion der Kripke-Struktur zu der Initialtransition der Aktionenfolge dieser Transition hinzugefügt wird und den Beginn des Lebenszyklus eines Objekts der Klasse *Bahnübergang* zeigt.

Nicht nur die Erreichbarkeit, sondern auch das Verlassen der Zustände ist wichtig. Eine Systemkomponente muss nicht nur jeden ihrer Zustände erreichen, sondern sie muss jeden erreichten Zustand auch verlassen können. Damit ein Objekt der Klasse *Bahnübergang* den Sicherungsvorgang ordnungsgemäß ausführt und abschließt, muss es jeden erreichten Zustand auch verlassen können und die nächsten Schritte des Sicherungsvorgangs ausführen. Beispielsweise, nach der Aktion *quittieren*(), die infolge des fahrzeugseitigen Stellbefehls *element_anfordern*(*bü*) ausgeführt wird, muss die Aktion *lzA_einschalten*() ausgeführt werden. Diese Bedingung entspricht der Sicherheitsanforderung, die das zuverlässige Starten des Sicherungsvorgangs eines Bahnübergangs nach dem Erhalt eines Stellbefehls sicherstellt. Diese Bedingung wird mit der temporallogischen Formel „**AG** (*quittieren*() ⇒ **AF** *lzA_einschalten*())“ ausgedrückt. Diese Formel verlangt, dass auf allen Pfaden, die von *quittieren*() ausgehen, *lzA_einschalten*() vorkommen muss.

Weitere Verifikationsziele sind die Überprüfungen bezüglich der funktionalen Anforderungen, die mit Hilfe von Sequenzdiagrammen modelliert worden sind, oder bezüglich der sicherheitsrelevanten Anforderungen, die unter Systemeigenschaften in Abschnitt 3.2.2 genannt wurden. Für diese Fälle wurden Beispiele in Kapitel sieben gegeben.

8.3 Durchführung des Model-Checking

Bei der formalen Verifikation lieferte der Model-Checker SMV beim ersten Modellentwurf für die Prüfbedingung „**AG** (*quittieren*() ⇒ **AF** *lzA_einschalten*())“ einen Fehlerpfad. Dieser Pfad führt nach der Ausführung der Aktion *quittieren*() über die Transition, die durch das Ereignis *außerBetriebNehmen*(*bü*) gesteuert wird, in den Endzustand. Damit ließ das Model zu, dass die Objekte der Klasse *Bahnübergang* nach der Ausführung der Aktion *quittieren*() ihre Aktivitäten beenden, ohne den quittierten Stellbefehl zu bearbeiten und ohne den Sicherungsvorgang abzuschließen. Folglich ging der Bahnübergang in den Endzustand über, während ein Stellbefehl vorliegt, und das Fahrzeug sich auf dem Weg zum Bahnübergang befand. Diese fehlerhafte Stelle ist in Abb. 8.1 eingekreist worden.

8.4 Identifikation von Fehlerquellen im Modell

In Kapitel drei wurde gezeigt, dass die Klasse *Bahnübergang* eine Spezialisierung der Klasse *Fahrwegelement* ist. Dabei erbt die Klasse *Bahnübergang* sowohl die statischen als auch die dynamischen Eigenschaften der Generalisierungsklasse *Fahrwegelement*.

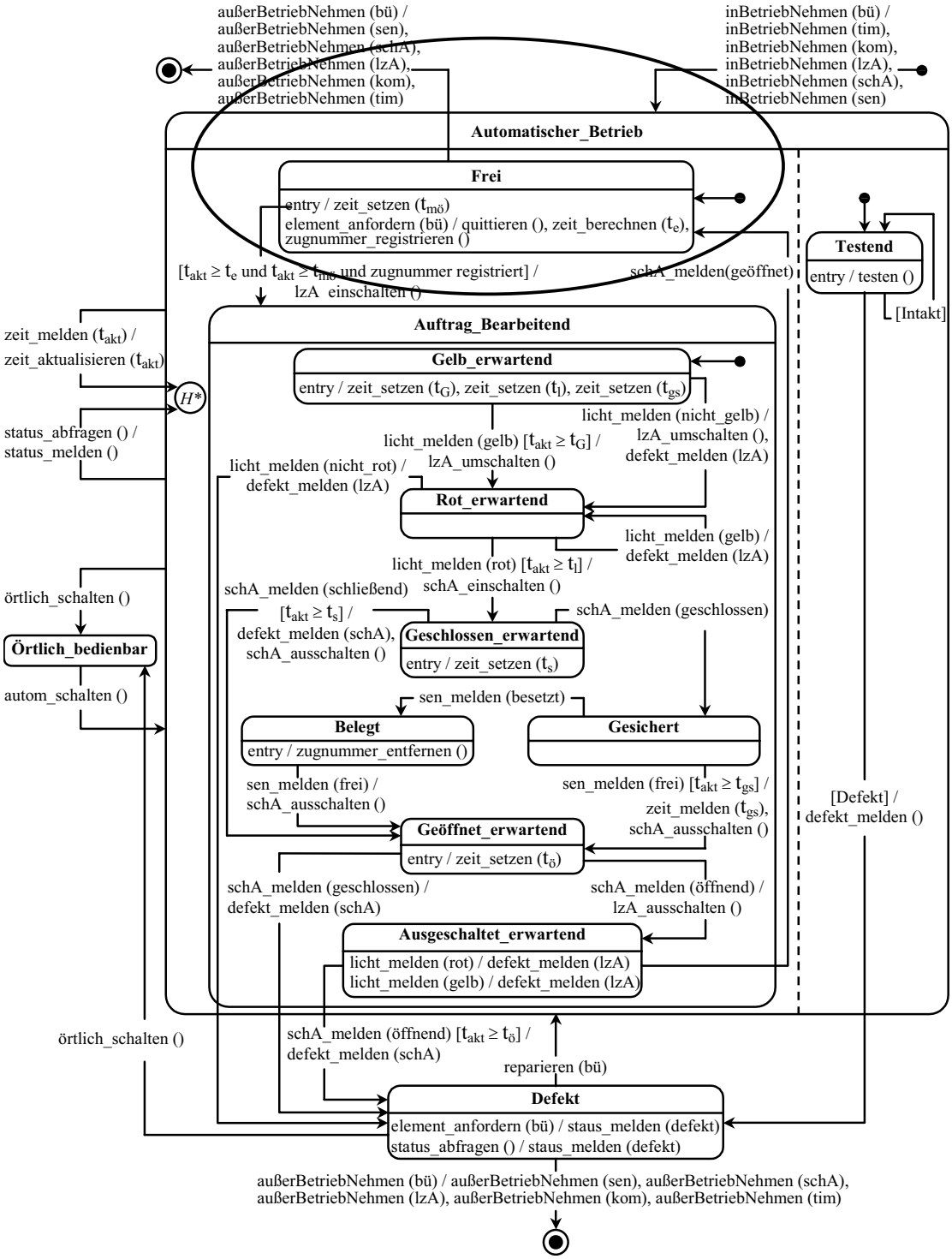


Abb. 8.1 Der geerbte Fehler in der Klasse Bahnübergang

Nach Definitionen 6.4.3 und 6.4.4.5.1 sind die Zustände der Spezialisierungsklasse entweder die von der Generalisierungsklasse geerbten Zustände oder ihre Erweiterun-

gen. Daraus folgt, dass jeder in der Klasse *Bahnübergang* entdeckte Fehler auch in der Klasse *Fahrwegelement* vorhanden sein kann.

In dem Zustandsdiagramm der Klasse *Bahnübergang* ist die Aktion *quittieren*() in dem Teilzustand *Frei* definiert worden, der aus der Klasse *Fahrwegelement* geerbt worden ist. Dieser Teilzustand kann auch in der Klasse *Fahrwegelement* mit dem Ereignis *außerBetriebNehmen* (*bü*) durch eine Transition verlassen werden, die in den Endzustand führt.

In diesem Fall betrifft die Korrektur des Modells nicht nur die Klasse *Bahnübergang*, sondern auch die Generalisierungsklasse *Fahrwegelement*. Abb. 8.2 zeigt den genannten Fehler in dem Zustandsdiagramm der Klasse *Fahrwegelement*.

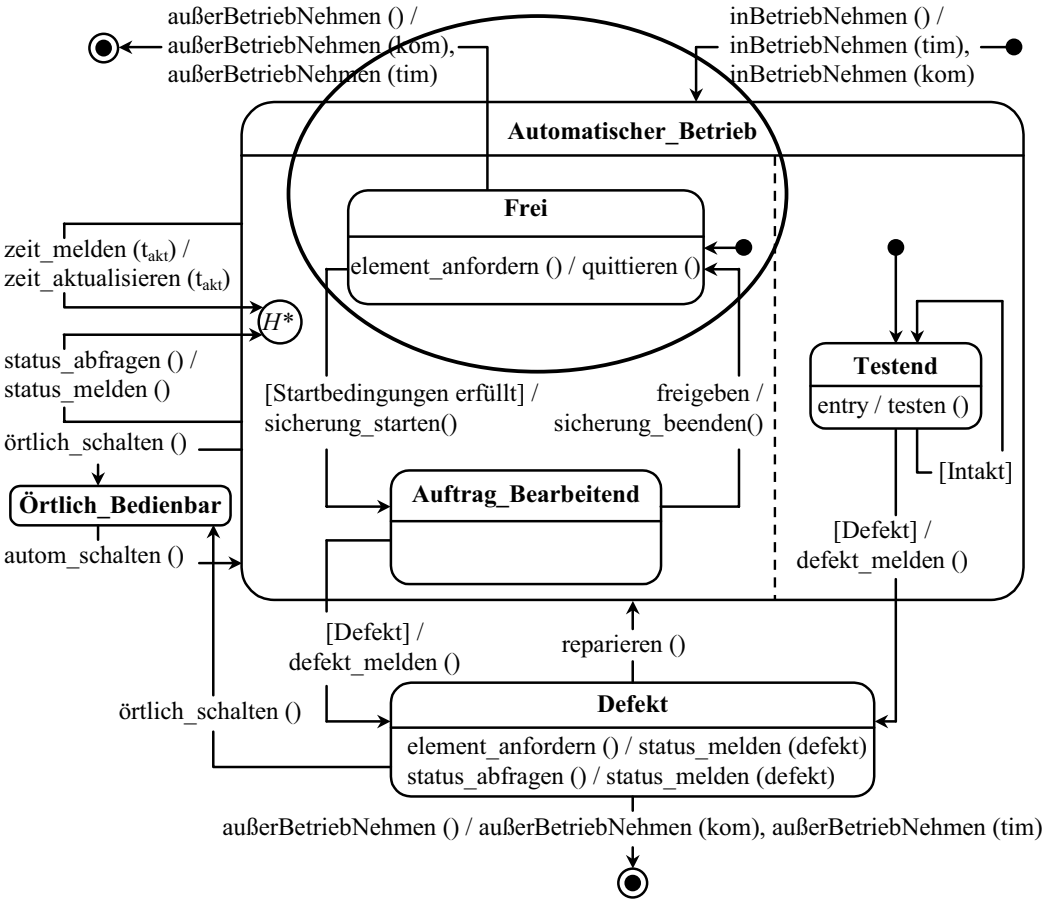


Abb. 8.2 Der vererbte Fehler in der Klasse Fahrwegelement

In dem Zustandsdiagramm der Klasse *Bahnübergang* kann der Teilzustand *Rot_erwartend* durch zwei Ereignisse *licht_melden*(*nicht_rot*) bzw. *licht_melden*(*gelb*) verlassen werden, die in Abb. 8.3 eingekreist worden sind. Da diese Ereignisse gleichzeitig eintreten können, sind für diesen Zustand zwei konkurrierende Austrittsbedingungen definiert worden. Diese Austrittsbedingungen steuern zwei Transitionen mit Zielzuständen *Defekt* bzw. *Rot_erwartend*.

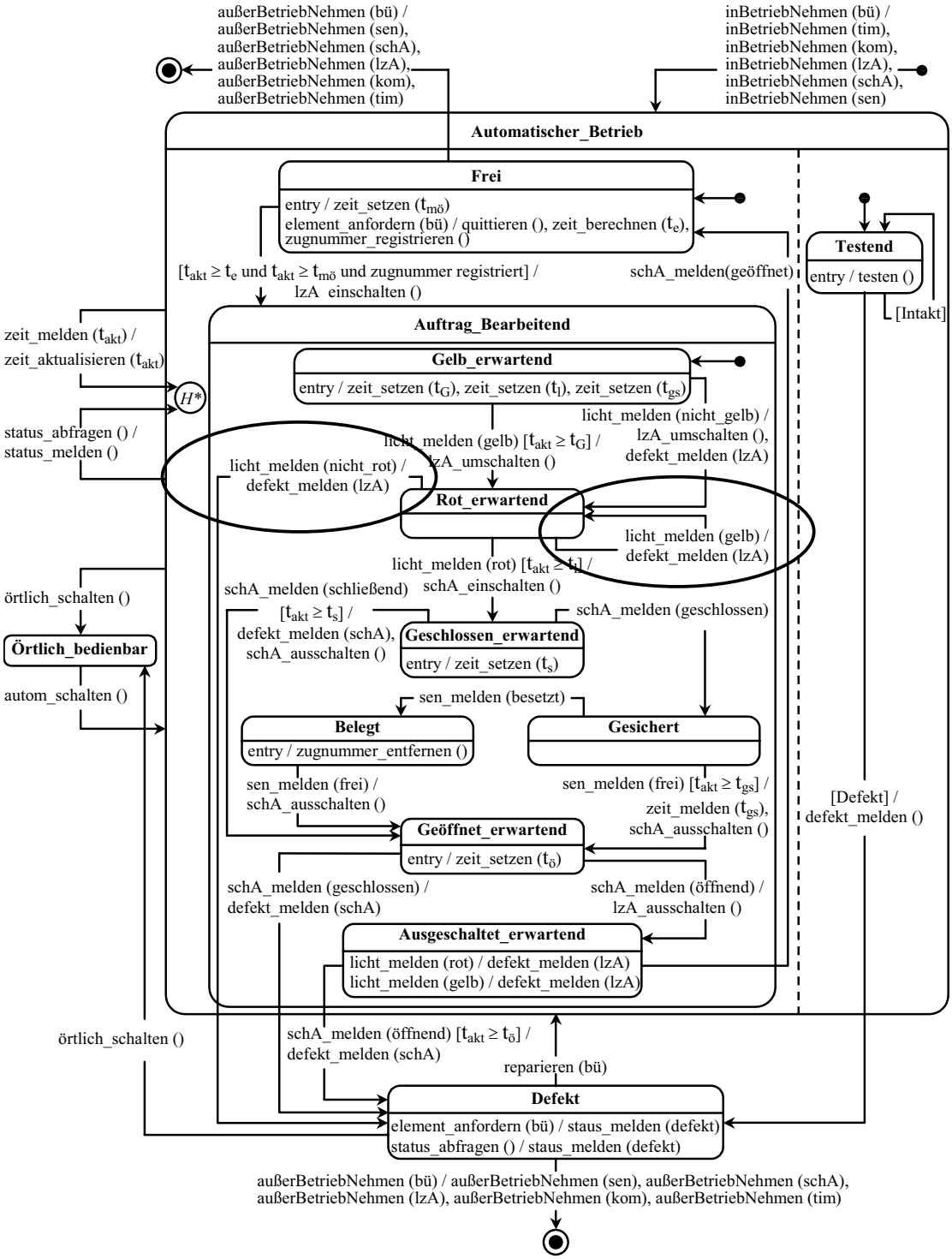


Abb. 8.3 Konkurrierende Austrittsbedingungen in der Klasse Bahnübergang

Nach Regel 5.5.2 dürfen konkurrierende Austrittsbedingungen nur dann für einen Zustand definiert werden, wenn sie Transitionen mit Eintritt in den gleichen Zustand oder in Zustände steuern, deren Oberzustände parallele Regionen eines dritten Zustands sind.

Die Zielzustände *Defekt* und *Rot_erwartend* erfüllen diese Bedingungen nicht. Dieser Fall stellt einen weiteren Fehler in der Klasse *Bahnübergang* dar, der zu einem ungültigen Verhalten der Objekte dieser Klasse führen kann. Dieser Fehler äußert sich bei dem nicht über die Vererbung übertragen Zustand *Rot_erwartend*. Daher bedarf die Klasse *Fahrwegelement* keiner Korrektur hinsichtlich dieses Fehlers.

8.5 Korrektur des Modells

Der erste in Abschnitt 8.3 genannte Fehler lässt sich durch die Einführung eines neuen Zustands beheben, so dass die Ausführung der Aktion *quittieren*() und die Reaktion auf das Ereignis *außerBetriebNehmen*(*bü*) in verschiedenen Zuständen erfolgen. Der neue Zustand wird nun für die Aktivitäten vorgesehen, die nach dem Erhalt eines Stellbefehls durchgeführt werden. Dieser Zustand wird *Reserviert* genannt und in das Zustandsdiagramm der Klasse *Fahrwegelement* eingeführt. Der alte Zustand *Frei* modelliert nun die Aktivitäten vor dem Erhalt eines Stellbefehls. In diesem Zustand kann nun auf das Ereignis *außerBetriebNehmen*(*bü*) entsprechend reagiert werden. Die Änderung des Zustandsdiagramms der Klasse *Fahrwegelement* wird in Abb. 8.4 gezeigt.

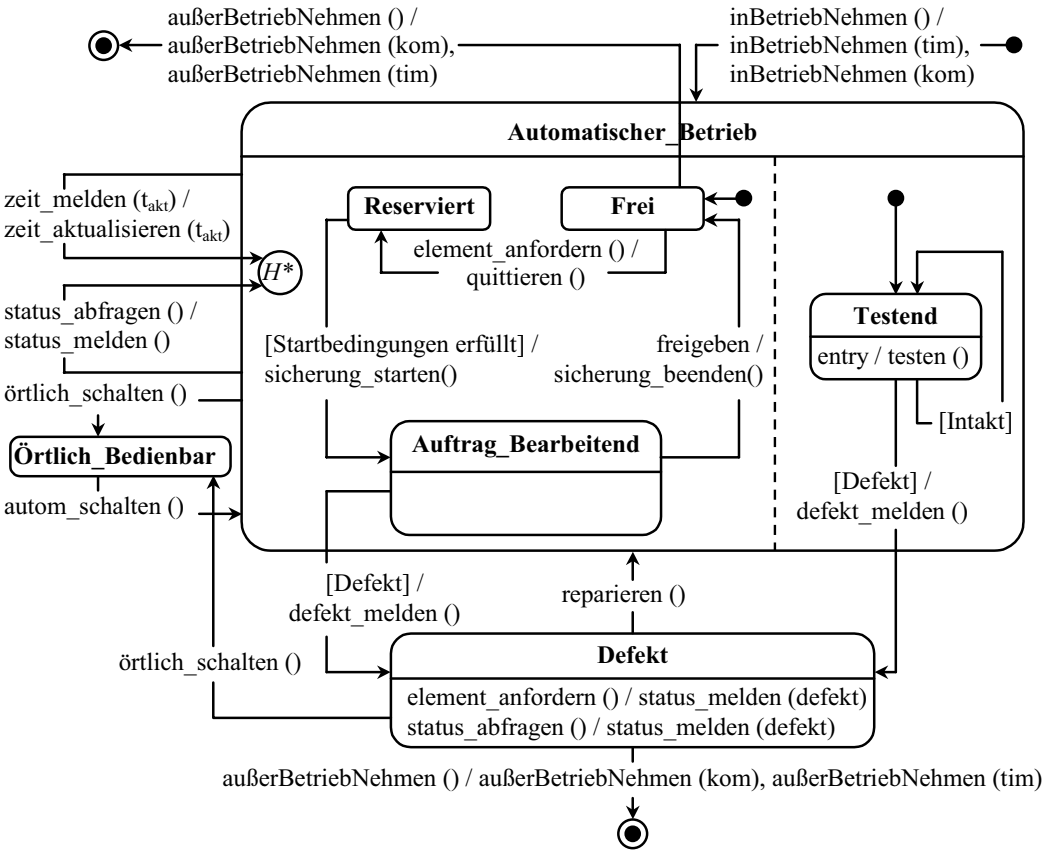


Abb. 8.4 Korrektur des Zustandsdiagramms der Klasse Fahrwegelement

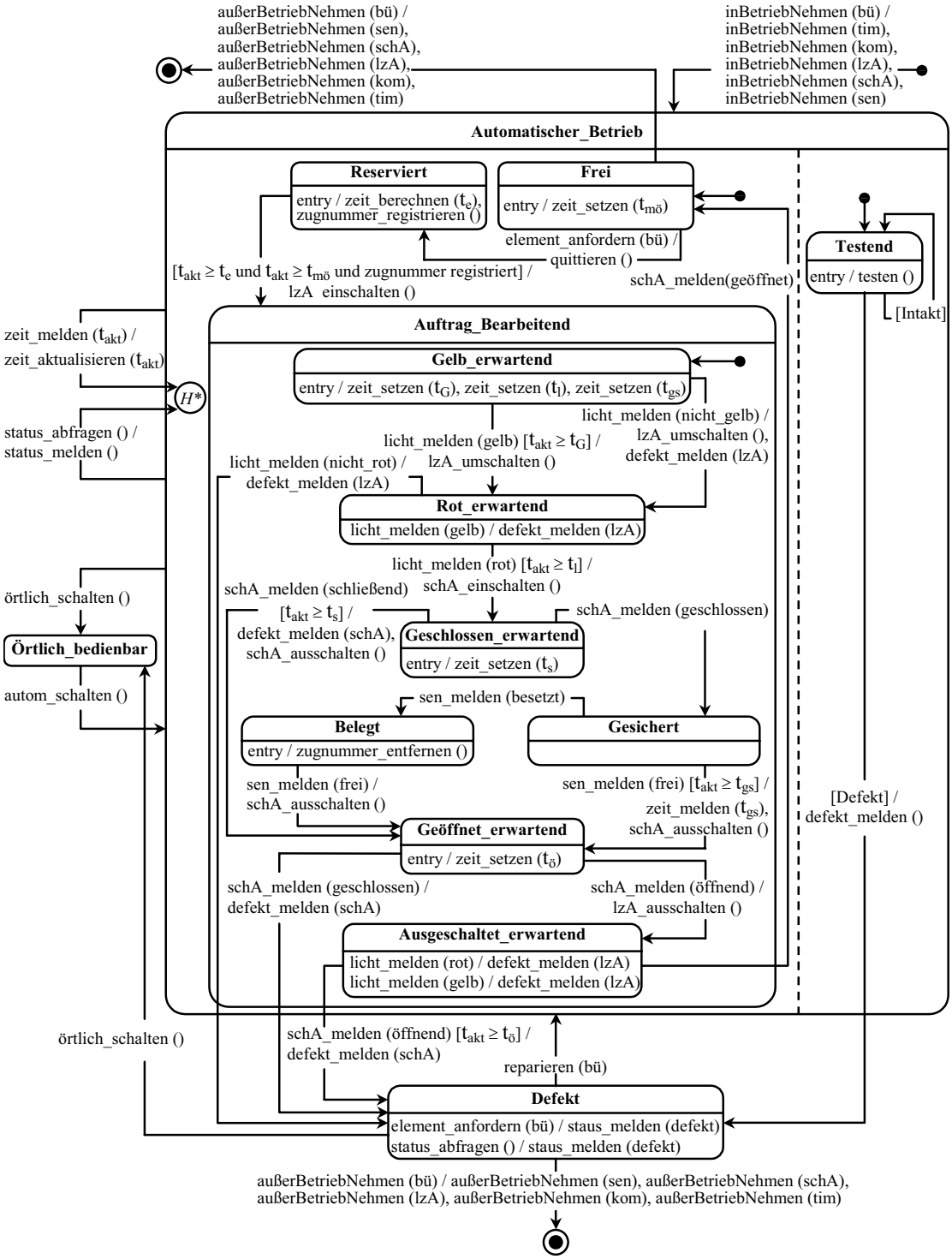


Abb. 8.5 Korrektur des Zustandsdiagramms der Klasse Bahnübergang

Die Zustände *Frei* und *Reserviert* werden durch die Vererbung auf die Spezialisierungsklasse *Bahnübergang* übertragen. In dieser Klasse wird in dem Zustand *Frei* die Aktion *zeit_setzen* ($t_{mö}$) ausgeführt, die für die Überwachung der Mindestöffnungszeit

zwischen zwei Sicherungsvorgängen vorgesehen ist. Mit dem Ereignis *element_anfordern*(*bü*) wird der Zustand *Frei* verlassen und die Aktion *quittieren*() ausgeführt. Danach befindet sich der Bahnübergang in dem Zustand *Reserviert*. In diesem Zustand wird die Aktionenfolge *zeit_berechnen*(t_e), *zugnummer_registrieren*(), die nach dem Erhalt eines Stellbefehls auszuführen ist, ausgeführt.

Der Fehler mit den konkurrierenden Austrittsbedingungen lässt sich durch die Änderung der Transition beseitigen, die von dem Ereignis *licht_melden*(*gelb*) gesteuert wird. Die Aktion *defekt_melden*(*lzA*), die bei dieser Transition ausgeführt wird, wird nun als interne Aktion des Zustands *Rot_erwartend* definiert. Jetzt ist für den Zustand *Rot_erwartend* das Ereignis *licht_melden*(*nicht_rot*) eine Austrittsbedingung, während *licht_melden*(*gelb*) eine Ausführungsbedingung einer internen Aktion ist. Gemäß Regel 5.5.3 hat das Ereignis *licht_melden* (*nicht_rot*) eine höhere Priorität, wenn es zur gleichen Zeit mit dem Ereignis *licht_melden* (*gelb*) eintritt. Dadurch wird ein ungültiges Verhalten der Objekte der Klasse *Bahnübergang* vermieden. In der Abb. 8.5 wird die Korrektur des Zustandsdiagramms der Klasse *Bahnübergang* dargestellt.

Beim Model-Checking ist zu beachten, dass der Model-Checker für jede Prüfbedingung den Pfad ausgibt, in dem die Prüfbedingung zum ersten Mal verletzt wird. Das bedeutet, dass nach der Korrektur der fehlerhaften Stelle im Modell das Model-Checking wieder durchgeführt werden muss, um weitere mögliche Fehler zu entdecken.

Wie am Anfang dieses Kapitels erwähnt wurde, können die hier präsentierten Fehler in einer textuellen Systemanforderungsspezifikation nur sehr schwer identifiziert werden. Die automatische Überprüfung einer formal fundierten Systemanforderungsspezifikation erleichtert die Fehlersuche und verhindert die Fortpflanzung von Fehlern in den weiteren Entwicklungsphasen.

9 Zusammenfassung und Ausblick

In der vorliegenden Arbeit ist ein Weg zum Einsatz von formalen Methoden für die Spezifikation und die Verifikation von funktionalen Anforderungen an die Systeme der Leit- und Sicherungstechnik im Eisenbahnwesen aufgezeigt worden. Eine direkte Anwendung von formalen Methoden erschwert den Ingenieuren im Eisenbahnwesen die Erstellung und die notwendige Korrektur von Systemanforderungsspezifikationen. Daher wurden hier grafische Beschreibungsmittel verwendet, für die eine formale Fundierung zur Verfügung gestellt worden ist. Die vorgelegte formale Semantik ermöglicht einerseits eine präzise Systemdefinition und andererseits eine formale Verifikation der Systemspezifikation.

Der hier präsentierte Ansatz basiert auf dem objektorientierten Paradigma. Demnach sind die Systemkomponenten Objekte eines Objektsystems, die miteinander und mit der Umgebung kommunizieren. Der objektorientierte Ansatz bietet die Möglichkeit, ein umfassendes und kompliziertes System zu analysieren und die Implikationen von Systemkomponenten zu untersuchen. Die Betrachtung von Entitäten als gekapselte und interagierende Einheiten entspricht der Sichtweise von Ingenieuren, die Systemkomponenten als modular und interaktiv ansehen.

Die Systeme der Eisenbahnsicherungstechnik lassen sich als Objektsysteme modellieren, in denen die Kommunikation unter den Systemkomponenten ereignisgesteuert und abhängig von Bedingungen stattfindet. Um die Grundelemente dieser Systeme festzule-

gen, wurden hier die Begriffe Objekt, Klasse, Objektsystem, Aktion, Ereignis und Bedingung definiert.

Die Spezifikation der Systemanforderungen konzentriert sich in dieser Arbeit auf die funktionalen Anforderungen an das System, die in die Beschreibung der statischen Struktur und des dynamischen Verhaltens des Systems abgebildet werden. Die Spezifikation dieser Systemeigenschaften erfolgt durch eine Modellbildung mit Hilfe einer Auswahl von grafischen Notationen der Unified Modeling Language (UML). In diesem Modell sind die Systemstruktur durch die Klassendiagramme, das dynamische Verhalten der einzelnen Systemkomponenten durch die Zustandsdiagramme und die Interaktionen zwischen den Komponenten durch die Sequenzdiagramme der UML spezifiziert worden.

Die angewendeten Diagrammarten der UML ermöglichen zwar eine kompakte, übersichtliche und nachvollziehbare Systembeschreibung, sie sind wegen mangelnder formaler Semantik jedoch nicht eindeutig und interpretationsanfällig. Dieses Problem führt dazu, dass einerseits die Systemeigenschaften nicht präzise genug spezifiziert werden können, und andererseits die Systemspezifikation nicht formal verifizierbar ist.

In der vorliegenden Arbeit wurden auf Basis der Charakteristika der Eisenbahnsicherungssysteme und ausgehend von intuitiver Semantik der UML für die genannten Diagrammarten und für die darin enthaltenen Konstrukte formale, eindeutige und interpretationsfreie Definitionen eingeführt, die einerseits eine präzise und formal fundierte Systemanforderungsspezifikation erlauben und andererseits die Möglichkeit zur Durchführung einer formalen und automatischen Überprüfung der Systemspezifikation auf Konsistenz und auf Sicherheitskonformität bieten.

Bei den Zustandsdiagrammen der UML, die zur Spezifikation des lokalen Verhaltens einzelner Systemkomponenten verwendet worden sind, wurden die Begriffe Aktionen, Aktionenfolgen, entry-actions, do-actions, exit-actions, einfache, sequenzielle, parallele, hierarchische Zustände, History-States und Deep-History-States formal definiert. Weiter wurden die Aktivierung und das Verlassen der genannten Zustände, die infolge von Transitionen stattfinden, untersucht und präzisiert. Durch die Einführung von Prioritätsregeln zwischen den Ausführungsbedingungen von Aktionen bzw. Austrittsbedingungen von Zuständen wurde ein ungültiges und mehrdeutiges Verhalten von Systemkomponenten unterbunden.

In einem weiteren Teil befasst sich diese Arbeit mit der Beschreibung der Systemstruktur, die durch die Klassendiagramme der UML spezifiziert worden ist. In den Klassendiagrammen wurden für die Relationen Assoziation, Aggregation, Komposition und Vererbung formale Definitionen eingeführt, welche die statischen Beziehungen unter den Systemkomponenten spezifizieren. Bei diesen Definitionen wurden die ontologischen Merkmale des Systems unter Berücksichtigung des dynamischen Verhaltens von Systemkomponenten zugrunde gelegt. Die Assoziationen spezifizieren die Existenz einer Kommunikationsbeziehung unter den Komponenten. Die Relationen Aggregation

bzw. Komposition bezeichnen eine allgemeine bzw. eine existenzabhängige und exklusive Zusammengehörigkeit unter den Systemkomponenten. Dadurch wurde die Beziehung zwischen dem Systemverhalten und der Systemstruktur dargestellt. Auf Basis der eingeführten Definitionen ließen sich Konsistenzbedingungen zwischen den Spezifikationen der Systemstruktur, des dynamischen Verhaltens von einzelnen Systemkomponenten und der Interaktionen innerhalb des Systems bestimmen. Die Beziehung zwischen der statischen Struktur und dem dynamischen Verhalten des Systems wurde bei der Relation Vererbung dadurch abgebildet, dass bei dieser Relation nicht nur die Attribute und die Aktionen, wie in den meisten Werken über den objektorientierten Ansatz dargestellt wird, sondern alle Eigenschaften von Objekten der Generalisierungsklasse vererbt werden. In dieser Interpretation werden alle Zustände, die die Objekte der Generalisierungsklasse in ihrem Lebenszyklus einnehmen und alle Transitionen zwischen diesen Zuständen von der Spezialisierungsklasse geerbt. Demzufolge werden auch alle Relationen und alle Komponenten der Generalisierungsklasse an die Spezialisierungsklasse übertragen. Dadurch lassen sich verschiedene Sicherungselemente in unterschiedlichen Ausführungen adäquat spezifizieren.

Für die Spezifikation der Interaktionen unter den Systemkomponenten wurden die Sequenzdiagramme der UML verwendet. Für diese Diagrammarten wurde ebenfalls eine formale Definition eingeführt. Mit Hilfe dieser Definition lassen sich die Anforderungen, die in Sequenzdiagrammen spezifiziert werden, in die anschließende formale Verifikation einbinden.

Die formale Verifikation einer Systemanforderungsspezifikation wurde in dieser Arbeit durch die Anwendung der Technik Model-Checking durchgeführt. Im Gegensatz zum geläufigen Testen, in dem nur ein beschränkter Teil des Modells untersucht wird, werden beim Model-Checking alle möglichen Kombinationen von Zuständen der Systemkomponenten überprüft. Zu diesem Zweck wurde das Verifikationswerkzeug SMV als Model-Checker eingesetzt. SMV kann eine Kripke-Struktur, die aus einfachen Zuständen und Transitionen besteht, gegenüber temporallogischen Ausdrücken automatisch überprüfen. Damit das in der Notation der UML vorliegende Modell formal verifiziert werden kann, müssen die hierarchischen und parallelen Zustandsstrukturen in geeignete Kripke-Strukturen transformiert werden.

In der vorliegenden Arbeit wurden für die Notationen in den Zustandsdiagrammen der UML Transformationsschemata definiert, die eine modelltreue Übersetzung der Zustandsdiagramme der UML in Kripke-Strukturen ermöglichen. Diese Schemata umfassen in erster Linie die Übertragung von Aktionen und Aktionenfolgen in geeignete Kripke-Strukturen. Durch die Anwendung von hier eingeführten Operationen sequentieller und paralleler Verknüpfung lassen sich aus den Kripke-Strukturen einzelner Aktionenfolgen entsprechende Kripke-Strukturen für verschiedene Anordnungen von Aktionenfolgen definieren. Weiter wurden geeignete Schemata für die Transformation von entry-actions, do-actions, exit-actions, einfachen, sequenziellen, parallelen und hierarchischen Zuständen eingeführt. Ferner wurde ein Verfahren zur Kombination von Zu-

ständen der parallelen Regionen und der Übertragung von Transitionen auf die kombinierten Zustände präsentiert, das eine modelltreue Abbildung von Zuständen dieser Regionen in Kripke-Strukturen ermöglicht.

Die Anwendbarkeit der präsentierten Beschreibungsmittel wurde am Beispiel des eingleisigen Bahnübergangs im Funkfahrbetrieb demonstriert. Durch das hier vorgelegte Konzept werden die Vorteile der grafischen Notationen der UML, wie Anwenderfreundlichkeit, leichte Erlernbarkeit, Handhabbarkeit, Simulierbarkeit und Wiederverwendbarkeit in den weiteren Entwicklungsphasen ausgenutzt und eine formale Verifikation in die Entwicklung der Systemanforderungsspezifikation eingebunden. Dadurch ist eine formale Überprüfung der Systemanforderungsspezifikation auf Konsistenz und Sicherheitskonformität gegeben. Das entspricht einem anwenderfreundlichen und ingenieurgerechten Einsatz von formalen Methoden, der die Präzision der Systemanforderungsspezifikation erhöht und ihre formale Überprüfung unterstützt. Dieses Konzept erfüllt die Forderungen der CENELEC-Normen für die sicherheitsrelevante Eisenbahnsignaltechnik bezüglich des Einsatzes von formalen Methoden.

Die Werkzeuge der UML können bereits von der Definitionsphase über die Designphase bis hin zur Implementierungsphase eingesetzt werden. Sie bilden eine Kette aus Werkzeugen für die Systemanalyse, die Systemsimulation und für die automatische Code-Generierung. Der Schritt zwischen der Modellierung und Modellüberprüfung kann durch die Einbindung der hier eingeführten Definitionen für die ausgewählten Diagramme und durch die Implementierung der präsentierten Transformationsschemata automatisiert werden.

Eine weitere Forderung der CENELEC-Normen ist die Durchführung einer Risikoanalyse, die ausgehend von der Systemanforderungsspezifikation stattfindet. Heute erfolgt die Risikoanalyse auf Basis der in natürlicher Sprache verfassten Systemspezifikationen, in denen aufgrund der textuellen Systembeschreibung die Systemeigenschaften nicht präzise genug spezifiziert werden. Eine Spezifikation der Systemanforderungen durch die hier präsentierten Beschreibungsmittel kann wegen der Kompaktheit, Eindeutigkeit und Nachvollziehbarkeit die Präzision der Risikoanalyse erhöhen. Die positive Auswirkung einer derartigen Systemanforderungsspezifikation auf die Durchführung einer Risikoanalyse kann in weiteren Arbeiten untersucht und gezeigt werden.

Literatur

[ABG04] Arabestani, S.; Bitsch, F.; Gayen, J.-T.
Precise Definition of the Single-Track Level Crossing in Radio-Based Operation in UML Notation and Specification of Safety Requirements.
H. Ehrig et al. (Eds.): INT 2004, LNCS 3147, pp. 119-144, 2004, Springer-Verlag, Berlin Heidelberg 2004.

[AC00] Antscher, M.; Coraiola, A.
GSM-R network for the high speed line Rome-Naples.
SIGNAL und DRAHT, (92)5/2000, S. 42-45.

[ADt97] ADtranz
Lastenhefte zum Funkfahrbetrieb(*FFB*).
DB AG, 1997.

[AG00] Arabestani, S.; Gayen, J.-T.
Prinzip der Vererbung bei der objektorientierten Analyse am Beispiel der funkbasierten Bahnübergangssteuerung.
Forms 2000 - Formale Techniken für die Eisenbahnsicherung, Fortschritt-Berichte VDI, Reihe 12, Nr. 441, 2000.

[Ant99] Antscher, M.
Einführung von GSM-R bei den europäischen Bahnen.
SIGNAL und DRAHT, (91)6/1999, S. 5-7.

[Arm99] Arms, J.-C.
FunkFahrBetrieb - Realisierung der Pilotstrecke Brackwede - Dissen/Bad Rothenfelde.
SIGNAL und DRAHT, (91)12/1999, S. 17-19.

[Ara02] Arabestani, S.
Relations in Object-Oriented Analysis.
In H. Ehrig, M. Grosse-Rhode (Eds): Proceedings of 2nd International Workshop on Software Specification Techniques - Satelite Event of ETAPS 2002, pages 37-47, Grenoble France, April 2002.

[Bal00] Balzert, H.
Lehrbuch Software-Technik Software-Entwicklung.
Spektrum Akademischer Verlag, 2. Auflage, Heidelberg / Berlin 2000.

[Bal99] Balzert, H.
Lehrbuch Grundlagen der Informatik Konzepte und Notationen in UML, Java und C++, Algorithmik und Software-Technik Anwendungen.
Spektrum Akademischer Verlag, Heidelberg.Berlin 1999.

[BB+01] Bérard, B.; Bidoit, M.; Finkel, A.; Laroussinie, F.; Petit, A.; Petrucci, L. ; Schnoebelen, Ph. ; McKenzie, P.
Systems and Software Verification, *Model-Checking Techniques and Tools*.
Springer-Verlag Berlin Heidelberg, 2001.

[BD+04] Brill, M.; Damm, W.; Klose, J.; Westphal, B.; Wittke, H.
Live Sequence Charts, An Introduction to Lines, Arrows, and Strange Boxes in the Context of Formal Verification.
H. Ehrig et al. (Eds.): INT 2004, LNCS 3147, pp. 374-399, 2004, Springer-Verlag, Berlin Heidelberg 2004.

[BE+94] Blaha, M.; Eddy, F.; Lorensen, W.; Premerlani, W.; Rumbaugh, J.
Objektorientiertes Modellieren und Entwerfen.
Carl Hanser Verlag München 1994.

[BG02] Bitsch, F.; Göhner, P.
Spezifikation von Sicherheitsanforderungen mit Safety-Patterns.
In Tagungsband „Software Engineering in der industriellen Praxis“, VDI-Bericht-Nr. 1666, Düsseldorf: VDI Verlag GmbH, 2002, S. 29-40.

[BH99] Barbier, F.; Henderson-Sellers, B.
Black and White Diamonds.
Proceedings of <<UML>>´99-The Unified Modeling Language. Beyound the Standard, Second International Conference, Lecture Notes in Computer Science 1723, Springer 1999.

[BHK04] Born, M.; Holz, E.; Kath, O.
Softwareentwicklung mit UML 2; *die >>neuen<< Entwicklungstechniken UML 2*, *MOF 2 und MDA*
Addison-Wesley Verlag, München, 2004.

[BHP00] Baer, A.; Haxthausen, A. E.; Peleska, J.
Towards Domain-Specific Formal Specification Languages for Railway Control Systems.
In E. Schnieder and U. becker (eds.): Proceedings of the 9th IFAC Symposium on Control in Transportation Systems 2000, June 13-15, 2000, Braunschweig, Germany, pp. 147-152.

[Bit01] Bitsch, F.
Safety Patterns - The Key to Formal Specification of Safety Requirements.
In Voges, U. (ed.): Proceedings of 20th International Conference, SAFECOMP 2001 - Computer Safety Reliability and Security, Berlin, Heidelberg: Springer-Verlag, LNCS 2187, S. 176-189. 2001.

[Bjø03] Bjørner, D.
Railways Systems: Towards a Domain Theory.
Technical report, Informatics and Mathematical Modelling, Building 322, Richard Petersens Plads, Technical University of Denmark, DK-2800 Kgs. Lyngby, Denmark, 2003.

[Bjø03a] Bjørner, D.
New Results and Trends in Formal Techniques for the Development of Software for Transportation Systems.
In FORMS2003: Symposium on Formal Methods for Railway Operation and Control Systems. Institut für Verkehrssicherheit und Automatisierungstechnik, Techn.Univ. of Braunschweig, Germany, 15-16 May 2003. Conf. held at Techn.Univ. of Budapest, Hungary. Editors: Tarnai, G. and Schnieder, E. Germany.

[BK+02] Broy, M.; Krüger, I.; Prenninger, W.; Sandner, R.
From Scenarios to Hierarchical Broadcasting Software Architectures using UML-RT.
In: International Journal of Software Engineering and Knowledge Engineering (IJSEKE) 12:4, Y. Deng (ed.), pp. 1 - 20, World Scientific, 2002.

[Boo97] Booch, G.
Objektorientierte Analyse und Design mit Praktischen Anwendungsbeispielen.
Addison-Wesley 1997.

[Bre96] Breymann, U.
C++ Eine Einführung, Studienbücher der Informatik.
Carl Hanser Verlag, München / Wien 1996.

[CE81a] Clarke, E. M.; Emerson, E. A.
Design and Synthesis of Synchronization Skeletons Using Branching Time Temporal Logic.
Lecture Notes in Computer Science. 131: 52-71, 1981.

[CE81b] Clarke, E. M.; Emerson, E. A.
Characterizing Correctness Properties of Parallel Programs Using Fixpoints.
Lecture Notes in Computer Science. 85: 169-181, 1981.

[CFJ93] Clarke, E. M.; Filkorn, T.; Jha, S.
Exploiting Symmetry in Temporal Logic Model Checking.
In C. Courcoubetis (Eds.), Proceedings of the 5th Workshop on Computer-Aided Verification, Lecture Notes in Computer Science, Band 697, pp. 450-462, 1993.

[CGL92] Clarke, E. M.; Grumberg, O.; Long D. E.
Model Checking and Abstraction.
In Proc. 18th Annual ACM Symposium on Principles of Programming Languages, 1992.

[CGM96] Canver, E.; Gayen, J.-T.; Moik, A.
Formale Entwicklung der Steuerungssoftware für eine elektrisch ortsbediente Weiche mit VSE.
Ulmer Informatik-Berichte, Nr. 96-01, 1996.

[CGP00] Clarke, E. M., Jr., Grumberg, O., Peled, D. A.
Model-Checking.
The MIT Press Cambridge, Massachusetts London, England 2000.

[CLM89] Clarke, E. M.; Long, D. E.; McMillan, K. L.
Compositional Model Checking.
In Proceedings for the 4th Annual Symposium on Principles of Programming Languages, pp. 343-362, 1989.

[Daw91] Dawes, J.
The VDM-SL Reference Guide.
London: UCL Press / Pitman Publishing, 1991.

[DDK99] Damm, W.; Döhmen, G. and Klose, J.
Secure decentralized control of railway crossings.
In S. Gnesi and D. Latella, editors, Fourth International ERCIM Workshop on Formal Methods in Industrial Critical Systems, pages 115 - 132, 1999.

[deR+01] de Roever, W.-P.; de Boer, F.; Hannemann, U.; Hooman, J.; Lakhnech, Y.; Zwiers, J.
Concurrency Verification, Introduction to Compositional and Noncompositional Methods.
Cambridge University Press, 2001.

[DGH02] Diethers, K.; Goltz, U.; Huhn, M.
Model Checking UML Statecharts with Time.
UML 2002, Workshop on Critical Systems Development with UML, September 2002.

[DH01] Damm, W.; Harel, D.
LSCs: Breathing Life into Message Sequence Charts.
In S. Gnesi and D. Latella (Eds.): Formal Methods in System Design, 19 (2) pp. 45-80, 2001, Kluwer Academic Publishers. Manufactured in The Netherlands.

[DH97] Denker, G.; Harte, P.
Troll - An Object Oriented Formal Method for Distributed Information System Design: Syntax and Pragmatics.
Abteilung Informationssysteme in der TU Braunschweig 1997.

[DK01] Damm, W.; Klose, J.
Verification of a Radio-based Signaling System Using the Statemate Verification Environment.
In S. Gnesi and D. Latella (Eds.): Formal Methods in System Design, 19 (2) 2001, Kluwer Academic Publishers. Manufactured in The Netherlands.

[DK+91] Duke, R.; King, P.; Rose, G.; Smith, G.
The Object-Z Specification Language.
Technical Report 91-1 (Version 1). Dept. of Computer Science, Software Verification Research Centre, University of Queensland, May 1991.

[dTd+96] de Boer, F.S.; Tej, H.; de Roever, W.-P.; van Hulst, M.
Compositionality in real-time shared-variable concurrency.
In Proc. of the 17th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS96), volume 1180 of LNCS. Springer-Verlag, 1996.

[Dür92] Dürr, E. H.
VDM++ - A Formal Specification Language for Object-Oriented Design.
In: Technology of Object-Oriented Languages and System. Proc of TOOLS Europe 1991, Hemel Hempstead: Prentice-Hall, 1992, pp 63-68.

[EC00] Ehrich, H.-D.; Caleiro, C.
Specifying communication in distributed information systems.
Acta Informatica, 2000.

[EC+98] Ehrich, H.-D.; Caleiro, C.; Sernadas, A.; Denker, G.
Logics for Specifying Concurrent Information Systems.
In Chomicki, J. and Saake, G., editors, Logics for Databases and Information Systems, pages 167-198. Kluwer Academic Publishers, 1998.

[EF+98] Evett, M.P.; France, R.B.; Larrondo-Petrie, M.M. and Saksena, M.
Extending Aggregation Constructs in UML.
The Unified Modeling Language, Select Papers of <<UML>>´98 Beyound the Notation, First International Workshop, Lecture Notes in Computer Science 1618, Springer 1998.

[Ehr99] Ehrich, H.-D.
Object Specification.
Astesiano, E.; Kreowski, H.-J. and Krieg-Brückner, B. editors. Algebraic Foundations of System Specification, chapter 12, pages 435-465. Springer-Verlag, Berlin 1999.

[EKP03] Ehrich, H.-D.; Kollmann, M.; Pinger, R.
Checking Object System Designs Incrementally.
Journal of Universal Computer Science, 9(2):106-119, February 2003.

[EKP02] Ehrich, H.-D.; Kollmann, M.; Pinger, R.
Distributed Model Checking.
In Haneberg, D. and Schellhorn, G. and Reif, W., editors, Proc. FM-TOOLS 2002, pages 53-58, Augsburg, 2002.

[EN50126] CELENEC EN 50126.
Railway Applications - The specification and demonstration of Reliability, Availability, Maintainability and Safety (RAMS), 1999.

[EN50128] CELENEC EN 50128.
Railway Applications - Communications, signaling and processing systems - Software for railway control and protection systems, 2001.

[EN50129] CELENEC prEN 50129.
Railway Apllications - Safety related electronic systems for signaling, 2000.

[EP00] Ehrich, H.-D.; Pinger, R.
Checking object systems via multiple observers.
In International ICSC Congresson Intelligent Systems & Applications (ISA'2000), volume 1, pages 242-248. University of Wollongong, Australia, International Computer Science Conventions (ICSC), Canada, 2000.

[ES99] Eberitsch, J.; Stöpel, U.
GSM-R-Terminals.
SIGNAL+DRAHT, (91)4/1999, S. 23-25.

[ES93] Emerson, E. A.; Sistla, A. P.
Symmetry and Model Checking.
In C. Courcoubetis (Eds.), Proceedings of the 5th Workshop on Computer-Aided Verification, Lecture Notes in Computer Science, Band 697, pp. 463-478, 1993.

[EW00] Eshuis, R., Wieringa, R. J.
Requirements-level semantics for UML statecharts.
In S. F. Smith and C. L. Talcott (Eds.): Formal Methods for Open Object-Based Distributed Systems IV, pp. 121-140, Boston, 2000, Kluwer Academic Publishers.

[FM01] Flake, S., Mueller, W.
An OCL Extension for Real-Time Constraints.
In T. Clark and J. Warmer (Eds.): Advances in Object Modelling with the OCL, Heidelberg: Springer-Verlag. 2001.

[FMR00] Flake, S.; Müller, W.; Ruf, J.
Structures English for Specification in Model Checking.
In Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen, S. 91-100. GI/ITG/GMM Workshop. Frankfurt. 2000.

[GL97] Godziejewski, B.; Lochmann, L.
Prinzipien der ERTMS/ETCS-Signalgebung auf Nebenbahnen.
SIGNAL und DRAHT, (89)12/1997, S. 14-15.

[GZ02] Gogolla, M.; Ziemann, P.
OCL Extended with Temporal Logic.
In Critical Systems Development with UML, pages 53-62. Technische Universität München, Institut für Informatik, 2002.

[Har87] D. Harel.
Statecharts: *A visual formalism for complex systems*.
Science of Computer Programming, 8:231–274, 1987.

[Hen99] Henning, S.
Diagnose an BÜSA in Verbindung mit Betriebszentralen und FFB.
SIGNAL und DRAHT, (91)12/1999, S. 39-41.

[Hen98] Henning, S.
Funktionalität von Bahnübergängen im FFB-Netz.
SIGNAL und DRAHT, (90)12/1998, S. 19-21.

[HK04] Harel, D.; Kugler, H.
The RHAPSODY Semantics of Statecharts (*or, On the Executable Core of the UML*).
H. Ehrig et al. (Eds.): INT 2004, LNCS 3147, pp. 325-354, 2004, Springer-Verlag, Berlin Heidelberg 2004.

[Hor98] Hornemann, K.
EUROBALISE zur Geschwindigkeitsüberwachung von NeiTech-Zügen.
SIGNAL und DRAHT, (90)12/1998, S. 37-41.

[Hol97] Holzmann, G. J.
The Model Checker SPIN.
IEEE Transactionsion Software Engineering, 23(5):279-295, 1997.

[HP98] Harel, D., Politi, M.
Modeling Reactive Systems with Statecharts The Statemate Approach.
McGraw-Hill 1998.

[HP03] Haxthausen, A. E.; Peleska, J.
Automatic Verification, Validation and Test for Railway Control Systems based on Domain-Specific Descriptions.
In Tsugawa, S. and Aoki, M. (eds): Proceedings of the 10th IFAC Symposium on Control in Transportation Systems. Elsevier Science Ltd, Oxford. ISBN 0-08-044059-2 (2003).

[HP03a] Haxthausen, A. E.; Peleska, J.
Generation of Executable Railway Control Components from Domain-Specific Descriptions.
In Tarnai, G. ; Hrsg.: Schnieder, E. (eds): Formal Methods for Railway Operation and Control Systems: Proceedings of Symposium FORMS'2003, Budapest/Hungary, May 15-16, Budapest: L'Harmattan Hongrie, 2003, pp.83-90.

[HP02] Haxthausen, A. E.; Peleska, J.
A domain specific language for railway control systems.
In Proceedings of the Sixth Biennial World Conference on Integrated Design and Process Technology, IDPT2002, Pasadena, California, June 23-28, 2002.

[HR00] Huth, M., Ryan, M.
Logic in Computer Science, *Modelling and reasoning about systems*.
Cambridge university press, 2000.

[Jon90] Jones, C. B.
Systematic Software Development Using VDM.
2nd Edition. Hemel Hempstead: Prentice-Hall, 1990.

[JW01] Jansen, D. N., Wieringa, R. J.
Techniques for Reactive System Design: The Tools in TRADE.
In K. R. Dittrich, A. Geppert and M. C. Norrie (Eds.): Advanced Information Systems Engineering: 13th international conference, CAiSE, LNCS 2068, p. 93 ff, Springer-Verlag Berlin Heidelberg 2001.

[Kat99] Katoen, J.-P.
Concepts, Algorithms, and Tools for Model Checking.
Bericht 1, Friedrich Alexander Universität Erlangen-Nürnberg, Juni 1999.

[Klo96] Klose, C.
Funkbasierter Fahrbetrieb (*FFB*).
SIGNAL und DRAHT, (88)12/1996, S. 29-30.

[KMR01] Knapp, A., Merz, S., Rauh, C.
Model Checking Timed UML State Machines and Collaborations.
Workshop on Software Model Checking, Electronic Notes in Theoretical Computer Science ENTCS volume 55 number 3, 2001.

[Kne99] Knewitz, R.
Aufbau des digitalen Mobilfunknetzes für den Bahnbetrieb.
SIGNAL und DRAHT, (91)11/1999, S. 33-34.

[Koc97] Koch, T.
Rechnergestützter Sicherheitsnachweis: Ein Verfahren zum Ausschluss gefährlicher Systemzustände in rechnergesteuerten Eisenbahn-Sicherungsanlagen.
Dresden: Technische Universität, Fakultät Verkehrswissenschaften „Friedrich List“, Dissertation 1997.

[KP88] Katz, S.; Peled, D.
An Efficient Verification Method for Parallel and Distributed Programs.
In Workshop on Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency, Lecture Notes in Computer Science, Band 354, pp. 489-507, 1988.

[Kuz04] Kuzmierz, A.
Überprüfung eines objektorientierten UML-Modells zum eingleisigen Bahnübergang im FFB mit Model-Checking.
Studienarbeit, Technische Universität Braunschweig, Institut für Software, Abteilung Informationssysteme, Institut für Eisenbahnwesen und Verkehrssicherung, März 2004.

[Lan91] Lano, K.
Z++, an Object-Oriented Extension to Z.
In: Nicholls, J. (Ed.): Z User Meeting, Oxford, UK. Workshops in Computing. Berlin, Springer, 1991.

[Las98] Lasch, R.
GSM-R-Endgeräte für die Bahnen.
SIGNAL und DRAHT, (90)12/1998, S. 32-34.

[Lig02] Liggesmeyer, P.
Software-Qualität, *Testen*, *Analysieren und Verifizieren von Software*.
Spektrum Akademischer Verlag Heidelberg Berlin, 2002.

[LMM99a] Latella, D., Majsik, I., Massnik, M.
Towards A Formal Operational Semantics of UML Statechart Diagrams.
In Proc. FMOODS`99, Third International Conference on Formal Methods for Open Object-Based Distributed Systems, Italy 1999.

[LMM99b] Latella, D., Majsik, I., Massnik, M.
Automatic Verification of a Behavioural Subset of UML Statechart Diagrams Using the SPIN Model-Checker.
Formal Aspects Computing, volume 11 number 6: pp. 637-664, 1999.

[LP99] Lilius, J., Paltor, I. P.
The Semantics of UML State Machines.
Turku Centre for Computer Science TUCS Technical Report No 273, May 1999.

[LPW98] Larsen, K. G.; Pettersson, P.; Yi, W.
UPPAAL in a Nutshell.
In Int. Journal on Software Tools for Technology Transfer 1(1-2), pages 134 152. Springer-Verlag. 1998.

[LQV01] Lavazza, L.; Quaroni, G. and Venturelli, M.
Combining UML and Formal Notations for Modelling Real-Time Systems.
In 8th European Conference Software Engineering, Wien, 2001.

[McM96] McMillan, K. L.
Symbolic Model Checking.
Kluwer Academic Publishers, 2. Auflage, 1996.

[Oes99] Oestreich, B.
Objektorientierte Softwareentwicklung - Analyse und Design mit der Unified Modelling Language.
R. Oldenburg Verlag, München/Wien, 1999.

[OG76] Owicki, S.; Gries, D.
An Axiomatic Proof Technique for Parallel Programs.
Acta Informatica, 6, pp. 319-340, 1976.

[OMG04] OMG
UML 2.0 Superstructure Specification.
Document: ptc/04-10-02 (convenience document), October 8, 2004.

[OMG03] OMG
Unified Modeling Language Specification.
Version 1.5, OMG Document formal/03-03-01, March 2003.

[Ort97] Ortner, E.
Methodenneutraler Fachentwurf - Zu den Grundlagen einer anwendungsorientierten Informatik.
Stuttgart, Leipzig: B.G. Teubner Verlagsgesellschaft, 1997.

[Pac04] Pachl, J.
Systemtechnik des Schienenverkehrs.
4. Aufl., Verlag B.G. Teubner 2004.

[Par98] Partsch, H.
Requirerements-Engineering systematisch, Modellbildung für Software-gestützte Systeme.
Berlin, Heidelberg, New York: Springer 1998.

[Paz00] Pazzi, L.
Part-Whole Statecharts for the Explicit Representation of Compound Behaviours.
Proceedings of <<UML>> 2000-The Unified Modeling Language Advancing the Standard, Third International Conference, Lecture Notes in Computer Science 1939, Springer 2000.

[Pet62] Petri, C. A.
Kommunikation mit Automaten.
Universität Bonn, Institut für Instrumentelle Mathematik, Dissertation, 1962.

[Pet81] Peterson, J. L.
Petri Net Theory and the Modelling of Systems.
Englewood Cliffs, N. J., Prentice-Hall, 1981.

[Pin02] Pinger, R.
Kompositionale Verifikation nebenläufiger Softwaremodelle durch Model Checking.
PhD thesis, Technical University Braunschweig, Germany, February 2002.

[QS81] Quielle, J. P.; Sifakis, J.
Specification and Verification of Concurrent Systems in CESAR.
In Proceedings of the Fifth International Symposium in Programming, 1981.

[RE97] Reißner, F.; Ebel, J.
Funkfahrbetrieb – technisches Konzept.
SIGNAL und DRAHT, (89)9/1997, S. 28-34.

[Rum96] Rumpe, B.
Formale Methodik des Entwurfs verteilter objektorientierter Systeme.
Dissertation an der Fakultät für Informatik der Technischen Universität München, 1996.

[Sch97] Schienmann, B.
Objektorientierter Fachentwurf.
Stuttgart, Leipzig: B. G. Teubner Verlagsgesellschaft. 1997.

[Sco04] Scott, K.
Fast Track UML 2.0
Springer-Verlag, Heidelberg, 2004.

[Spi92] Spivey, J. M.
The Z Notation.
A Reference Manual. 2nd Edition. New York: Prentice-Hall, 1992.

[Ste01] Stevens, P.
On Associations in the Unified Modelling Language.
In Gogolla, M. and Kobryn,C. (Eds.): UML 2001, LNCS 2185, pp. 361–375, Springer-Verlag Berlin Heidelberg 2001.

[UML04] <<UML>> 2004.
The Unified Modeling Language.
7th International Conference, Baar, T.; et al. (Eds.), LNCS 3273, Springer-Verlag, 2004.

[UML03] <<UML>> 2003.
The Unified Modeling Language.
6th International Conference, Stevens, P.; Whittle, J.; Booch, G. (Eds.), LNCS 2863, Springer-Verlag, 2003.

[UML02] <<UML>> 2002.
The Unified Modeling Language.
5th International Conference, Jezequel, J.-M.; Hussmann, H.; Cook, S. (Eds.), LNCS 2460, Springer-Verlag, 2002.

[UML01] <<UML>> 2001.
The Unified Modeling Language.
4th International Conference, Gogolla, M.; Kobryn, C. (Eds.), LNCS 2185, Springer-Verlag, 2001.

[UML00] <<UML>> 2000.
The Unified Modeling Language.
Third International Conference, Evans, A.; Kent, S.; Selic, B. (Eds.), LNCS 1939, Springer-Verlag, 2000.

[UML99] <<UML>> 1999.
The Unified Modeling Language.
Second International Conference, France, R.; Rumpe, R. (Eds.), LNCS 1723. Springer-Verlag, 1999.

[UML98] <<UML>> 1998.
The Unified Modeling Language.
First International Workshop, Bezivin, J.; Muller, P.-A. (Eds.), LNCS 1618. Springer-Verlag, 1998.

[Val90] Valmari, A.
A Stubborn Attack on State Explosion.
In Computer Aided Veri_cation, Lecture Notes in Computer Science, Band 531, pp. 156-165, 1990.

[VdB01] Von der Beeck, M.
Formalization of UML-Statecharts.
M. Gogolla, C. Kobryn (Eds.): UML 2001, LNCS 2185, pp. 406-421. Springer, Berlin Heidelberg 2001.

[Waw99] Wawrzyński, W.
Models of control systems in transport with regarding of reliability aspects. Safety and Reliability, Edited by Schuëller, G.I. & Kafka, P., A.A.Balkema Rotterdam Brookfield 1999.

[Wit99] Wittmann, K.
GSM-R Applikationen.
SIGNAL und DRAHT, (91)4/1999, S. 26-27.

Tabelle der Symbole und Abkürzungen

Symbol	Bedeutung
$\bullet$	Symbol, das mittels eines Pfeils auf den Anfangszustand zeigt
$\circledcirc$	Symbol zur Bezeichnung eines Endzustands
$\subseteq$	Teilmenge
$\in$	Element von
$\cup$	Vereinigung von Mengen
$\cap$	Durchschnitt von Mengen
$\setminus$	Differenz von Mengen
$\forall$	Für alle
$\exists$	Es existiert
$\mid$	Oder
$\wedge$	Konjunktion (und)
$\vee$	Disjunktion (oder)
$\neg$	Verneinung (nicht)
$\Rightarrow$	Implikation (folgt)
$\varnothing$	Leere Menge
$\oplus$	Sequentielle Verknüpfung von Kripke-Strukturen
$\otimes$	Parallele Verknüpfung von Kripke-Strukturen

$\|\|$	Parallele Zustände
$\nparallel$	nichtparallele Zustände
$\preccurlyeq$	Ordnungsrelation in einem Sequenzdiagramm
$\cong$	Isomorphie
a_{ε}	Leere Aktion
a_{fin}	Finalaktion
a_{init}	Initialaktion
$\mathcal{A}_{K^p}$	Menge der Aktionen von Objekten der Klasse K^p
$\mathcal{A}_O$	Menge der Aktionen des Objekts O
$act(z)$	Eine Aktionenfolge im Zustand z
$\mathcal{A}ttr_{K^p}$	Menge der Attribute von Objekten der Klasse K^p
$\mathcal{A}ttr_O$	Menge der Attribute des Objekts O
$\mathcal{A}ttr_O^{dy}$	Menge der dynamischen Attribute des Objekts O
$\mathcal{A}ttr_O^{st}$	Menge der statischen Attribute des Objekts O
$\mathcal{A}_{K^q} \rightarrow \mathcal{A}_{K^p}$	Abbildung einer Menge auf eine andere Menge
$\mapsto$	Abbildung eines Elements auf ein anderes
$z_i \rightarrow z_j$	Transition von z_i nach z_j
$\mathcal{B}_O$	Menge der Bedingungen
$default()$	Initialzustand
$default(z)$	Initialzustand als Teilzustand von z
$do(z)$	Do-actions des Zustands z
e_{ε}	Leeres Ereignis
e_{fin}	Finalereignis
e_{init}	Initialereignis
$\mathcal{E}_{K^p}$	Menge der für die Objekte der Klasse K^p definierten Ereignisse
$\mathcal{E}_O$	Menge der für das Objekt O definierten Ereignisse
$en(z)$	entry-actions des Zustands z
$ex(z)$	exit-actions des Zustands z
$final()$	Endzustand
$final(z)$	Endzustand als Teilzustand von z
$in(z)$	Menge aller Teilzustände von z
$include(z)$	Menge aller direkten Teilzustände von z
K^p	Klasse K^p
$K_{O_i}^p$	Objekt O_i der Klasse K^p
KS	Kripke-Struktur
$KS, s \vDash \phi$	Die Bedingung ϕ gilt in dem Zustand s der Struktur *Kripke-Struktur*
$KS, s \nvDash \phi$	Die Bedingung ϕ gilt nicht in dem Zustand s der Struktur *Kripke-Struktur*

M	Menge aller Objekte in einem Objektsystem
$\mathcal{M}_{K^i}$	Zustandsmaschine der Klasse K^i
$\mathbb{N}$	Menge der natürlichen Zahlen {1, 2, ... }
O	Objekt O
O_i	Objekt O_i
$\mathcal{P}(V)$	Menge aller Teilmengen von V
R	Menge der Anforderungen in einem Objektsystem
$\mathcal{R}_O$	Menge von Relationen von O
T	Transition
$\mathcal{T}$	Menge der Transitionen
V	Menge der atomaren Aussagen
$\mathcal{Z}$	Menge der Zustände

Index

هر چه گفتیم جز حکایت دوست
در همه عمر از آن پشیمانیم

سعدی

Lebenslauf

Persönliche Daten

Name:	Saeid Arabestani
Geburtsdatum:	31. Januar 1962
Geburtsort:	Teheran / Iran

Berufliche und Qualifikation

1971 - 1977:	Gymnasium in Teheran / Abschluss im Zweig Mathematik
1977 - 1979:	Pädagogische Hochschule in Teheran / Abschluss im Zweig Mathematik
1979 - 1987:	Lehrer für Mathematik in Teheraner Gymnasien
1989 - 1990:	Studienkolleg in Bochum / Abschluss Juni 1990
1990 - 1998:	Informatikstudium an der Technischen Universität Braunschweig
1998:	Abschluss Diplom-Informatiker
1998 - 2004:	Wissenschaftlicher Mitarbeiter am Institut für Eisenbahnwesen und Verkehrssicherung (IfEV), Technische Universität Braunschweig
2004:	Safety Manager bei Siemens AG Transportation Systems, Braunschweig

Veröffentlichungen

- Arabestani, Saeid; Bitsch, Friedemann; Gayen, Jan-Tecker
  Precise Definition of the Single-Track Level Crossing in Radio-Based Operation in UML Notation and Specification of Safety Requirements
  In H. Ehrig et al. (Eds.): INT 2004, LNCS 3147, pp. 119-144, Springer-Verlag 2004
- Arabestani, Saeid
  Umschreibung von Zustandsdiagrammen der UML für das Model-Checking
  In E. Schnieder (Hrsg.): Entwurfsmethodik, Modellbildung, Werkzeuge und Anwendungen, 8. Fachtagung Entwurf komplexer Automatisierungs-systeme EKA 2003, S. 63-79, Juni 2003, Technische Universität Braunschweig
- Arabestani, Saeid
  Relations in Object-Oriented Analysis
  In H. Ehrig, M. Grosse-Rhode (Eds): Proceedings of 2nd International Workshop on Software Specification Techniques - Satelite Event of ETAPS 2002, pages 37-47, Grenoble France, April 2002
- Arabestani, Saeid; Gayen, Jan-Tecker
  Bedeutung von Relationen in der objektorientierten Analyse
  Signal + Draht, 93(2001)12, S. 33-37, 2001
- Arabestani, Saeid; Gayen, Jan-Tecker
  Prinzip der Vererbung bei der objektorientierten Analyse am Beispiel der funkbasierten Bahnübergangssteuerung
  In E. Schnieder (Hrsg.): Forms 2000 - Formale Techniken für die Eisenbahnsicherung, Fortschritt-Berichte VDI, Reihe 12, Nr. 441, 2000, Technische Universität Braunschweig
- Arabestani, Saeid; Gayen, Jan-Tecker
  Objektorientierte Analyse zur Modellierung im Eisenbahnwesen
  Signal + Draht, 92(2000)1+2, S. 20-27
- Arabestani, Saeid; Gayen, Jan-Tecker
  Ein Weg zur Einsetzbarkeit formaler Methoden für Ingenieure
  In E. Schnieder (Hrsg.): Forms ´99 - Formale Techniken für die Eisenbahnsicherung, Fortschrittberichte VDI, Reihe 12, Nr. 436, 1999, Technische Universität Braunschweig

Braunschweig, August 2005

Veröffentlichungen der Schriftenreihe des IfEV

Schriftenreihe des Instituts für Eisenbahnwesen und Verkehrssicherung (IfEV) der Technischen Universität Carolo-Wilhelmina zu Braunschweig ISSN: 0721 - 7137

67 ARABESTANI, Saeid
Formal verifizierbare objektorientierte Systemspezifikationen mit UML für Eisenbahnsicherungssysteme
2005, 208 S., ISBN 3-923325-67-3

66 MICHAELSEN, Raimo
Quantitative Ermittlung und Bewertung des Risikos für Zugfahrten durch Flankenraumverletzungen
2004, 198 S., ISBN 3-923325-66-5

65 MASCHEK, Ulrich
Datenmodell zur Planung von Stellwerken
2002, 101 S., ISBN 3-923325-65-7

64 **Moderne Sicherheitsanalysen - Theorie und Praxis**
Vorträge, Kolloquium 8./9. Oktober 2001 in Braunschweig
2001, 137 S., ISBN 3-923325-64-9

63 **Verkehrsssysteme im Umbruch**
Referate, Praxisseminar 20./21. Oktober 1999 in Leipzig
1999, 180 S., ISBN 3-923325-63-0

62 MATZKE, Manfred
Ein Bewertungssystem für Betriebskonzepte bei Industriebahnen
1999, 196 S., ISBN 3-923325-62-2

61 MARTIN, Ullrich
Optimiertes Modell zur gewerkeübergreifenden Planung der Eisenbahninfrastruktur
1998, 117 S., ISBN 3-923325-61-4

60 Institut für Eisenbahnwesen und Verkehrssicherung (Hrsg.)
Beiträge aus dem Zentrum für Verkehr der Technischen Universität Braunschweig
1998, 118 S., ISBN 3-923325-60-6

59 FRERICHS, Michael
Verfahren zur Genauigkeitsbeurteilung GPS-gestützter Ortungsergebnisse bei Landfahrzeugen
1998, 98 S., ISBN 3-923325-59-2

58 PACHL, Jörn
Sicherheit im Eisenbahnverkehr
1997, 29 S., ISBN 3-923325-58-4

57 HINZ, Rainer
Verfahren zur Anpassung der Ortungsgenauigkeit an die Leistungsanforderung bei Eisenbahnstrecken
1997, 158 S., ISBN 3-923325-57-6

56 KRISTA, Matthias
Verfahren zur Fahrplanoptimierung dargestellt am Beispiel der Synchronzeiten
1996, 133 S., ISBN 3-923325-56-8

55 SIX, Jürgen
Seitliche Auslenkung von Fahrzeugen im Huckepackverkehr unter Sicherheitsgesichtspunkten
1996, 125 S., ISBN 3-923325-55-X

54 GLÖE, Günter
Ein Verfahren zum Nachweis des anforderungsgemäßen Zeitverhaltens von Rechnersytemen mit Sicherheitsverantwortung
1996, 229 S., ISBN 3-923325-54-1

53 DREIER, Johannes
Einsatz von selbstfahrenden Einzelwagen im Wagenladungsverkehr der Eisenbahn
1995, 201 S., ISBN 3-923325-53-3

52 MARTIN, Ulrich
Verfahren zur Bewertung von Zug- und Rangierfahrten bei der Disposition
1995, 211 S., ISBN 3-923325-52-5

51 ARMS, Jan-Christian
Strukturierte Analyse und Konzeptentwurf eines optimierten Prozeßleitsystems für Schienenbahnen
1994, 193 S., ISBN 3-923325-51-7

50 OLTROGGE, Christine
Linienplanung für mehrstufige Bedienungssysteme im öffentlichen Personenverkehr
1994, 187 S., ISBN 3-923325-50-9

49 PACHL, Jörn
Steuerlogik für Zuglenkanlagen zum Einsatz unter stochastischen Betriebsbedingungen
1993, 75 S., ISBN 3-923325-49-5

48 SCHEUNEMANN, Reinhard
Ersatz des Linienleiters durch Gleisstromkreise für Befehl und Meldung
1993, 173 S., ISBN 3-923325-48-7

47 FISCHER, Jan Paul
Expertensystemkonzept für die Fahrwegdisposition von Werksbahnen
1993, 68 S., ISBN 3-923325-47-9

46 MALINOWSKI, Uwe
Wissensbasierte Benutzerunterstützung für Software-Systeme entwickelt am Beispiel der Personaldisposition im Verkehr
1992, 135 S., ISBN 3-923325-46-0

45 LEHRACH, Karlheinz W.
Rechnergestützte Disposition bei gestörten Betriebsabläufen im Straßenbahnverkehr
1991, 185 S., ISBN 3-923325-45-2

44 PASTERNOK, Thomas
Selbstkonfigurierendes dezentrales Steuerungssystem für Bahnen
1991, 176 S., ISBN 3-923325-44-4

43 OHM, Wilfried
Rechnergestützte Gestaltung von Vorschriften
1991, 263 S., ISBN 3-923325-43-6

42 GAYEN, Jan-Tecker und KUCHTA, Dorota
Strukturorientierte Tests - Möglichkeiten und Grenzen der Fehleroffenbarung
1991, 95 S., ISBN 3-923325-42-8

41 Institut für Verkehr, Eisenbahnwesen und Verkehrssicherung
Fachseminar "Sicherheitstechnik am Beispiel des Schienenverkehrs - Entwicklung ins nächste Jahrtausend" 18./19. Oktober 1988 in Braunschweig
1988, 296 S., ISBN 3-923325-41-X

40 Institut für Verkehr, Eisenbahnwesen und Verkehrssicherung
Fachseminar "Systemregelung im öffentlichen Verkehr und bei den Eisenbahnen" 17./18. Februar 1988 in Braunschweig
1988, 202 S., ISBN 3-923325-40-1

39 EYLERT, Bernhard
Bewertung von Systemen zur Ortung und Navigation im Landverkehr
1989, 177 S., ISBN 3-923325-39-8

38 KESSLER, Wolfgang
Kapazitätsreserven von öffentlichen Personenverkehrsangeboten
1988, 167 S., ISBN 3-923325-38-X

37 FENGLER, Wolfgang
Rechnergestütztes Verfahren zur netzweiten Bestimmung der Abfahrlagen von Durchgangsgüterzügen (Simulationsmodell Dg-Netz)
1987, 202 S., ISBN 3-923325-37-1

36 SCHUCK, Helmut
Analoger Fensterkomparator in Fail-Safe-Technik
1987, 151 S., ISBN 3-923325-36-3

35 GROTTKER, Ulrich
Die Gefährdungswahrscheinlichkeit als Sicherheitskennwert technischer Systeme am Beispiel des Eisenbahnbetriebes in Abhängigkeit der Verspätungsverteilungen von Zugfahrten
1986, 190 S., ISBN 3-923325-35-5

34 SCHLÜTER, Eckhard
Personaleinsatzplanung im öffentlichen Personennahverkehr mit Hilfe interaktiver Rechentechnik
1986, 162 S., ISBN 3-923325-34-7

33 KLIMMEK, Dietmar
Modell für die Bewertung von Navigationssystemen im Landverkehr
1985, 105 S., ISBN 3-923325-33-9

32 BERTRAM, Hans-Henning
Interaktives Planungssystem für den öffentlichen Personennahverkehr - Ein Beitrag zur Optimierung von Wagenumläufen
1984, 145 S., ISBN 3-923325-32-0

31 GLIMM, Jochen
Über die sichere Längsregelung von Fahrzeugen und die Sicherheitsbewertung von Fahrzeugkolonnen
1983, 253 S.

30 **apl. Prof. Dr.-Ing. Hans FRICKE zum 70. Geburtstag**
1983, 237 S., ISBN 3-923325-30-4

29 FRICKE, Ullrich
Nachbildung der räumlich-zeitlichen Nachfrageverteilung zur Linienplanung im öffentlichen Personennahverkehr
1983, 120 S., ISBN 3-923325-29-0

28 BELING, Rainer
ÖPNV-Aufkommen eines Busnetzes im ländlichen Raum
1983, 180 S., ISBN 3-923325-28-2

27 GRABAND, Michael
Sicherheitsrelevante Funkdatenübertragung mit selbstsynchronisierendem Code für Verkehrssysteme
1982, 201 S., ISBN 3-923325-27-4

26 CLAUSEN, Andreas
Die Netzredundanz im Rahmen eines mehrstufigen Beschreibungsansatzes für Verkehrsnetze
1982, 228 S., ISBN 3-923325-26-6

25 REINHARDT, Winfried
Gefährdungswahrscheinlichkeit in einem realen spurgeführten Verkehrssystem
1982, 184 S., ISBN 3-923325-25-8

24 WEIGAND, Werner
Graphentheoretisches Verfahren zur Fahrplangestaltung in Transportnetzen unter Berücksichtigung von Pufferzeiten mittels interaktiver Rechentechnik
1981, 257 S., ISBN 3-923325-24-X

23 KRAFT, Karl Heinz
Zugverspätungen und Betriebssteuerung von Stadtschnellbahnen in systemtheoretischer Analyse
1981, 200 S., ISBN 3-923325-23-1

22 TCHINDA, Albert
Energieoptimales automatisches Nahverkehrssystem mit Mikroprozessor auf dem Fahrzeug
1980, 166 S., ISBN 3-923325-22-3

21 ECKLUNDT, Hinrich
Modellbewertung der digitalen Simulation eines DLS-geführten Landeanfluges im Interferenzfeld
1979, 146 S., ISBN 3-923325-21-5

20 BELING, Rainer
nl **Untersuchungen über die Einsatzmöglichkeiten und Grenzen des Schienenverkehrs als Zubringer zu Flughäfen**
1979, 259 S., ISBN 3-923325-20-7

19 POTTGIESSER, Hans
Die Entstehung und Entwicklung der Hauptsignale der Deutschen Eisenbahnen
1979, 147 S., ISBN 3-923325-19-3

18 DIENST, Hartmut
Linienplanung im spurgeführten Personenfernverkehr mit Hilfe eines heuristischen Verfahrens
1978, 272 S., ISBN 3-923325-18-5

17 GAYEN, Jan-Tecker
Die Gefährdungswahrscheinlichkeit in einem spurgebundenen Verkehrssystem unter Berücksichtigung des Verkehrsprozesses
1978, 115 S., ISBN 3-923325-17-7

16 KIRSCH, Jochen
Die Abstandszielbremsung - ein Verfahren zur Steuerung von Talbremsen in Ablaufanlagen
1978, 191 S., ISBN 3-923325-16-9

15 WOJANOWSKI, Erich
Linienförmiges Zugsicherungs- und Zugsteuerungssystem auf der Grundlage äquidistanter Gleisstromkreise mit Dezentralisierung der Streckenausrüstung
1978, 167 S., ISBN 3-923325-15-0

14 SELLE, Dieter
Mehrfachleitungstheorie und Mehrtortheorie zur Berechnung von Linienleitern im Gleis
1977, 79 S., 3-923325-14-2

13 SCHILDT, Gerhard-Helge
Zeitsynchrones Kommunikationssystem mit zentraler Synchronisation durch einen Normalfrequenzsender für ein bedarfsgesteuertes Verkehrssystem
1977, 140 S., ISBN 3-923325-13-4

12 **Lehrstuhl und Institut für Verkehr, Eisenbahnwesen und Verkehrssicherung 1951-1976**
1976, 232 S., ISBN 3-923325-12-6

11 AL-HIJAJ, Abdul-Halim
Funktionsanalyse von Stellwerken durch Programmablaufpläne und ihre Umsetzung in Logikpläne
1976, 96 S., ISBN 3-923325-11-8

10 WEHNER, Ludwig
Methode zur Erfassung und Auswertung der Zuverlässigkeit der Signaltechnik
1976, 117 S., ISBN 3-923325-10-X

9 WIEGAND, Klaus-Dieter
Optimierung der Regel- und Bedarfstransporte im spurgeführten Güterfernverkehr unter Berücksichtigung von Transportketten
1976, 222 S., ISBN 3-923325-09-6

8 WEGEL, Helmut
Fahrplangestaltung für taktbetriebene Nahverkehrsnetze
1974, 187 S., ISBN 3-923325-08-8

7 SCHNEIDER, Wolfgang
Berechnung der Sicherheit von parallelredundanten Schaltwerken
1974, 201 S., ISBN 3-923325-07-X

6 GÖBERTSHAHN, Rudolf
Mehrgruppenbildung im ersten Verteilgang mit Hilfe der Fischgräten-Gleisanordnung aus betriebstechnischer und ökonomischer Sicht
1974, 174 S., ISBN 3-923325-06-1

5 PAGEL, Ernst-Olav
Anwendungsmmöglichkeiten hochfrequenter Gleisstromkreise
1973, 101 S., ISBN 3-923325-05-3

4 GÜNTER, Horst
Wirtschaftspolitische Beurteilung staatlicher Lenkungsmaßnahmen im Binnengüterverkehr
1972, 143 S., ISBN 3-923325-04-5

3 GLIMM, Jochen
Fastoptimale Abstandregelung von Schienenfahrzeugen mit Hilfe elektronisch simulierter Leitfahrzeuge
1972, 209 S., ISBN 3-923325-03-7

2 SPIESS, Peter
Betriebsprogramme für punktförmige Zugbeeinflussungsanlagen
1972, 111 S., ISBN 3-923325-02-9

1 PANNEK, Gernot
Simulation von Signalsystemen zur Abstandshaltung von Schienenfahrzeugen unter besonderer Berücksichtigung digitaler Weginformation
1971, 154 S., ISBN 3-923325-01-0

0 ISENSEE, Alfred
Der Einfluß verschiedener Betriebsarten und Parameter auf das Verhalten isolierstoßloser Gleisstromkreise bei höheren Frequenzen
1970, 286 S. (keine ISBN-Nr.)

Digitaldruck und Einband . Buchbinderei Wolfram Schmidt . Braunschweig
Telefon: 0531/33 75 89 . E-Mail: Schmidt.Druck.BS@t-online.de

ISSN 0721-7137
ISBN 3-923325-67-3